Non-Deterministic Abstract Machines

Malgorzata Biernacka, Dariusz Biernacki, Serguei Lenglet, Alan Schmitt

Non-Deterministic Abstract Machines

Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, Alan Schmitt

RESEARCH REPORT
N° 9475
July 2022
Project-Team Epicure

ISSN 0249-6399 ISRN INRIA/RR--9475--FR+ENG

Non-Deterministic Abstract Machines

Małgorzata Biernacka[*], Dariusz Biernacki[†], Sergueï Lenglet[‡], Alan Schmitt[§]

Project-Team Epicure

Research Report n° 9475 — July 2022 — 33 pages

Abstract: We present a generic design of abstract machines for non-deterministic programming languages, such as process calculi or concurrent lambda calculi, that provides a simple way to implement them. Such a machine traverses a term in the search for a redex, making non-deterministic choices when several paths are possible and backtracking when it reaches a dead end, i.e., an irreducible subterm. The search is guaranteed to terminate thanks to term annotations the machine introduces along the way.

We show how to automatically derive a non-deterministic abstract machine from a zipper semantics—a form of structural operational semantics in which the decomposition process of a term into a context and a redex is made explicit. The derivation method ensures the soundness and completeness of the machines w.r.t. the zipper semantics.

Key-words: λ-calculus, process calculi, abstract machines

[*] University of Wrocław, Wrocław, Poland
[†] University of Wrocław, Wrocław, Poland
[‡] Université de Lorraine, Nancy, France
[§] INRIA, Rennes, France

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Machines abstraites non déterministes

Résumé : Nous proposons une présentation uniforme des machines abstraites pour les langages non déterministes, tels que les calculs de processus ou les lambda-calculs concurrents, qui permet de les implémenter facilement. Une telle machine traverse le terme à la recherche d'un redex, en faisant des choix arbitraires lorsque plusieurs chemins sont possibles, et en retournant en arrière lorsqu'elle atteint un cul-de-sac, c'est-à-dire un terme irreductible. Nous garantissons la terminaison de la recherche grâce aux annotations que la machine ajoute en cours de route.

Nous montrons comment dériver automatiquement une machine non déterministe depuis une sémantique *zipper*—une forme de sémantique opérationnelle structurelle dans laquelle la décomposition d'un terme en un contexte et un redex apparaît explicitement. La méthode de dérivation garantit la correction et la complétude de la machine par rapport à la sémantique zipper.

Mots-clés : λ-calcul, calculs de processus, machines abstraites

2012 ACM Subject Classification Theory of computation → Abstract machines

Keywords and phrases Abstract machines, non-determinism, lambda-calculus, process calculi

1 Introduction

Abstract machines, i.e., first-order tail-recursive transition systems for term reduction, such as SECD [28], CEK [12], and the KAM [27], are a traditional and celebrated artifact in the area of programming languages based on the λ-calculus. They serve both as a form of operational semantics [11, 12, 28] and an implementation model [25, 32] of programming languages, but they also play a role in other areas, e.g., in proof theory [27], higher-order model checking [40], or cost models [1]. They are used as an implementation model also in concurrent languages [15, 17, 33, 36, 45], in particular to study distribution [4, 18–20, 23, 37].

Since in general designing a new abstract machine is a serious undertaking, several frameworks supporting mechanical or even automatic derivations of abstract machines from other forms of semantics have been developed [2, 6, 22, 43]. However, these frameworks assume a language that satisfies the unique decomposition property [6, 10], which entails that at each step one specific redex is selected, and thus the language follows a deterministic reduction strategy. This property does not hold in non-deterministic languages such as process calculi (or even in the λ-calculus without a fixed reduction strategy) and the existing methodology cannot be applied. Existing machines for non-deterministic languages are ad-hoc and may not be complete, i.e., not all reduction paths of the language can be simulated by the corresponding abstract machine [15, 17, 19, 33, 45].

This work presents a generic framework for the definition of complete abstract machines that implement a non-deterministic reduction relation in a systematic way. The idea is to go through a term to find a redex without following a specific strategy, picking arbitrarily a subterm when several are available—e.g., going left or right of an application in λ-calculus. The two main ideas are: (1) the machine should not remain stuck when it chooses a subterm which cannot reduce—in such a case we make it backtrack to its last choice; (2) the machine should not endlessly loop searching for redexes in subterms which cannot reduce—the machine annotates the subterms which are normal forms to prevent itself from visiting them again.

Non-deterministic machines designed in this way can be complex even for small languages, therefore we show how to generate them automatically from an intermediary *zipper semantics*. This semantics, inspired by Huet [24], is a form of structural operational semantics (SOS) [39] that remembers the current position in a term by building a context, i.e., a syntactic object that represents a term with a hole [13]. This format of semantics makes it explicit how a term is decomposed into a context and a redex, and thus it can be seen as a non-deterministic counterpart of the decomposition function in (deterministic) context-based reduction semantics [9, 14]. While deterministic reduction semantics is directly implementable and the corresponding abstract machine can be viewed (roughly) as its optimization [10], non-deterministic reduction semantics, even when expressed as a zipper semantics, requires non-trivial instrumentation to become implementable in a complete way. Deriving the non-deterministic abstract machine (NDAM) from the zipper semantics consists exactly in such an instrumentation with the backtracking mechanism and normal-form annotations. We show how to derive an NDAM from an arbitrary zipper semantics that satisfies minimal conditions, and we prove that the resulting NDAM is sound and complete w.r.t. the semantics. Our approach applies in particular to process calculi, for which the abstract machines defined so far were ad-hoc and usually not complete.

$$\mathsf{init}\ \frac{t \xrightarrow{\bullet}_{\mathsf{app}} t'}{t \to_{\mathsf{zs}} t'} \qquad \mathsf{appL}\ \frac{t \xrightarrow{@\, s \,::\, \mathbb{E}}_{\mathsf{app}} t'}{t \mathbin{@} s \xrightarrow{\mathbb{E}}_{\mathsf{app}} t'} \qquad \mathsf{appR}\ \frac{s \xrightarrow{t\, @ \,::\, \mathbb{E}}_{\mathsf{app}} s'}{t \mathbin{@} s \xrightarrow{\mathbb{E}}_{\mathsf{app}} s'} \qquad \mathsf{app\lambda}\ \frac{t \xrightarrow{\lambda x \,::\, \mathbb{E}}_{\mathsf{app}} t'}{\lambda x.t \xrightarrow{\mathbb{E}}_{\mathsf{app}} t'} \qquad \mathsf{app\beta}\ \frac{t \xrightarrow{s, \mathbb{E}}_{\mathsf{lam}} t'}{t \mathbin{@} s \xrightarrow{\mathbb{E}}_{\mathsf{app}} t'}$$

$$\mathsf{lam\beta}\ \frac{}{\lambda x.t \xrightarrow{s, \mathbb{E}}_{\mathsf{lam}} \mathbb{E}[t\{s/x\}]}$$

Figure 1 Zipper semantics for the λ-calculus

The contributions of this paper are: (1) a generic design of sound and complete, non-deterministic abstract machines which cannot get stuck or infinitely loop in a redex search, (2) with a systematic derivation procedure from an intermediary format, called zipper semantics. The resulting machine is an implementation of the non-deterministic source language.

We illustrate our method on the λ-calculus without a fixed reduction strategy and on a minimal process calculus HOcore [30], respectively in Sections 2 and 3. We then give a derivation procedure of an NDAM from an arbitrary zipper semantics in Section 4. We discuss related work in Section 5 and future work in Section 6. The appendix contains the proofs missing from the body of the paper and a further example: the zipper semantics and abstract machine of HOπ [41] that extends HOcore with name restriction. An implementation of the derivation procedure is also available [5].

2 Lambda-calculus

As a warm-up example, we present the zipper semantics and the corresponding NDAM for the λ-calculus with no fixed reduction strategy.

2.1 Syntax and Context-based Reduction Semantics

We let t, s range over λ-terms. We denote application with an explicit operator @ to annotate it later on. We represent a context $\mathbb{E}$ as a list of elementary contexts called *frames* $\mathfrak{F}$.

$$t, s ::= x \mid \lambda x.t \mid t \mathbin{@} s \qquad \mathfrak{F} ::= \lambda x \mid @\, t \mid t\, @ \qquad \mathbb{E}, \mathbb{F}, \mathbb{G} ::= \bullet \mid \mathfrak{F} :: \mathbb{E}$$

Because it is more convenient for the definition of the machine, we interpret contexts inside-out [11]: the head of the context is the innermost frame. The definition of plugging a term in a context $\mathbb{E}[t]$ is therefore as follows:

$$\bullet[t] \triangleq t \qquad (\lambda x :: \mathbb{E})[t] \triangleq \mathbb{E}[\lambda x.t] \qquad (@\, s :: \mathbb{E})[t] \triangleq \mathbb{E}[t \mathbin{@} s] \qquad (s\, @ :: \mathbb{E})[t] \triangleq \mathbb{E}[s \mathbin{@} t]$$

We write $t\{s/x\}$ for the capture-avoiding substitution of x by s in t, and define the context-based reduction semantics $\to_{\mathsf{rs}}$ of the λ-calculus by the following rule

$$\mathbb{E}[(\lambda x.t) \mathbin{@} s] \to_{\mathsf{rs}} \mathbb{E}[t\{s/x\}]$$

which can be read declaratively: if we find a redex in a context $\mathbb{E}$ built according to the given grammar of contexts, then we can reduce. This format of semantics does not make it apparent how to *decompose* a term to find a redex. On the other hand, structural operational semantics offers another common semantic format that makes it more explicit how to navigate in a term to find a redex, but it does not store the traversed path.

2.2 Zipper Semantics

A first step towards an abstract machine is to make explicit the step-by-step decomposition of a term into a context and a redex. To this end, we propose zipper semantics, a combination of SOS and reduction semantics. Like a regular SOS, a zipper semantics goes through a term looking for a redex using structural rules, except the current position in the term is made explicit with a context as in reduction semantics.

The zipper semantics for the λ-calculus is defined in Figure 1. It looks for a β-redex while constructing the surrounding context $\mathbb{E}$ at the same time. The decomposition happens in the rules appL, appR, and appλ, where we search for a redex by descending into the appropriate subterm of a given term. Each of these rules corresponds to a frame, with init initiating the search by setting the context to $\bullet$.

These rules actually look for the application at the root of the β-redex; checking that an application $t \mathbin{@} s$ is indeed a β-redex is done by the rule appβ. It relies on an auxiliary transition $t \xrightarrow{s,\mathbb{E}}_{\mathsf{lam}} t'$, which checks that its source is indeed a λ-abstraction. In that case, we can β-reduce with rule lamβ. We can see that computation only occurs in the axiom; the other rules are simply propagating the result unchanged.

One may wonder why we need the rules appβ and lamβ while a single axiom $(\lambda x.t) \mathbin{@} s \xrightarrow{\mathbb{E}}_{\mathsf{app}} \mathbb{E}[t\{s/x\}]$ is enough to recognize a β-redex. The reason is that we restrict ourselves to patterns discriminating only the head constructor of a term, to remain close to an abstract machine where the decomposition of a term occurs only one operator at a time.

We prove that the zipper semantics and reduction semantics coincide in Appendix A.

▸ **Example 1.** To illustrate further how to recognize a redex one operator at a time, suppose we restrict the argument of the β-redex to a value $v ::= x \mid \lambda x.t$, so that $\mathbb{E}[(\lambda x.t) \mathbin{@} v] \rightarrow_{\mathsf{rs}} \mathbb{E}[t\{v/x\}]$. In such a case, we would need an extra transition $s \xrightarrow{x,t,\mathbb{E}}_{\mathsf{v}} t'$, checking that s is a value. The rule lamβ would be replaced by the rule lamβ$^{\mathsf{v}}$ below.

$$\mathsf{lam}\beta^{\mathsf{v}}\ \frac{s \xrightarrow{x,t,\mathbb{E}}_{\mathsf{v}} t'}{\lambda x.t \xrightarrow{s,\mathbb{E}}_{\mathsf{lam}} t'} \qquad \mathsf{var}^{\mathsf{v}}\ \frac{}{y \xrightarrow{x,t,\mathbb{E}}_{\mathsf{v}} \mathbb{E}[t\{y/x\}]} \qquad \mathsf{lam}^{\mathsf{v}}\ \frac{}{\lambda y.s \xrightarrow{x,t,\mathbb{E}}_{\mathsf{v}} \mathbb{E}[t\{\lambda y.s/x\}]}$$

◂

2.3 Non-Deterministic Abstract Machine

Design principles. Zipper semantics describes how to decompose a term into a redex and a context, but it is not yet an implementation, as it does not explain what to do when several rules can be applied, like appL, appR, and appβ. The NDAM simply picks one of them, and backtracks if it reaches a dead-end. We present how we implement this backtracking and how it can be derived from the zipper rules, before giving the formal definition of the NDAM.

The decomposition at work in the zipper semantics rules can be turned into machine steps: we see the change of focus occurring in the source term between the conclusion and the premise. We introduce a machine mode for each transition kind (here, app and lam), and the rules appL, appR, and appβ are translated to the following *forward* machine steps, with | separating the term from the context:

$$\langle t \mathbin{@} s \mid \mathbb{E}\rangle_{\mathsf{app}} \mapsto \langle t \mid {@}\, s :: \mathbb{E}\rangle_{\mathsf{app}} \quad \langle t \mathbin{@} s \mid \mathbb{E}\rangle_{\mathsf{app}} \mapsto \langle s \mid t \mathbin{@} :: \mathbb{E}\rangle_{\mathsf{app}} \quad \langle t \mathbin{@} s \mid \mathbb{E}\rangle_{\mathsf{app}} \mapsto \langle t \mid s, \mathbb{E}\rangle_{\mathsf{lam}}$$

We see why interpreting the context inside-out is convenient: focusing on t in $\mathbb{E}[t \mathbin{@} s]$ amounts to pushing the frame ${@}\, s$ on top of $\mathbb{E}$. It is the same as decomposing the term as $({@}\, s :: \mathbb{E})[t]$: the innermost constructor becomes the topmost one in the context.

The resulting machine is non-deterministic as three different steps can be taken from the configuration $\langle t \mathbin{@} s \,|\, \mathbb{E}\rangle_{\mathsf{app}}$. Unlike typical deterministic machines, it does not implement a strategy and does not choose, e.g., to always go left of an application as in the KAM [27]. A consequence is that the machine can make a wrong choice, i.e., focus on a term which cannot reduce, like a variable. In such cases, we want the machine to backtrack to the last configuration for which a choice had to be made, and no further. To do so, we record the applied rules in a stack π. When we reach a term which cannot reduce, we switch to a backtracking mode (here, bapp) where we can "unapply" a rule.

$$\begin{aligned}\langle t \mathbin{@} s\,;\pi \,|\, \mathbb{E}\rangle_{\mathsf{app}} &\mapsto \langle t\,; \mathsf{appL} :: \pi \,|\, @\, s :: \mathbb{E}\rangle_{\mathsf{app}}\\ \langle x\,;\pi \,|\, \mathbb{E}\rangle_{\mathsf{app}} &\mapsto \langle \pi\,; x \,|\, \mathbb{E}\rangle_{\mathsf{bapp}}\\ \langle \mathsf{appL} :: \pi\,; t \,|\, @\, s :: \mathbb{E}\rangle_{\mathsf{bapp}} &\mapsto \langle t \mathbin{@} s\,;\pi \,|\, \mathbb{E}\rangle_{\mathsf{app}}\end{aligned}$$

The machine may try other rules on $t \mathbin{@} s$, e.g., to find a redex in s. However, it should not try appL again, as the backtracking step implies there is no redex in t. We refer to backtracking steps like the last one as *backward*, and to steps like the middle one as *switching*. The backward step is simply the reverse of the corresponding forward step.

We prevent the machine from choosing a previously explored path by annotating the root operator of an already tested subterm. An annotation $t \mathbin{@^{\mathsf{app}}} s$ means that $t \mathbin{@} s$ has already been tried for $\xrightarrow{\mathbb{E}}_{\mathsf{app}}$ transitions and is a normal form for it. Similarly, a term annotated lam is a normal form w.r.t. $\xrightarrow{s,\mathbb{E}}_{\mathsf{lam}}$ (it is not a λ-abstraction). A term can be annotated with both app and lam, for instance if it is a variable.

The machine can take a forward step only if the term in focus has not been already tested. For $t \mathbin{@} s$, we try appL (resp. appR) only if t (resp. s) is not annotated app, and appβ only if t is not annotated lam. If none of the steps applies because of the annotations, then all possible rules have been tried and $t \mathbin{@} s$ is a normal form for app: the machine backtracks and annotates the term accordingly. In what follows, Σ represents an annotation set.

$$\begin{aligned}&\langle x^{\Sigma}\,;\pi \,|\, \mathbb{E}\rangle_{\mathsf{app}} \mapsto \langle \pi\,; x^{\Sigma\cup\{\mathsf{app}\}} \,|\, \mathbb{E}\rangle_{\mathsf{bapp}}\\ &\langle t \mathbin{@^{\Sigma}} s\,;\pi \,|\, \mathbb{E}\rangle_{\mathsf{app}} \mapsto \langle \pi\,; t \mathbin{@^{\Sigma\cup\{\mathsf{app}\}}} s \,|\, \mathbb{E}\rangle_{\mathsf{bapp}} \text{ if no other step applies}\end{aligned}$$

Switching steps are of two kinds: either the language construct does not have a forward step for a given mode (like a variable in the app mode), or all possible rules have been tried for the construct. They both can be derived from the zipper semantics by looking at which rule can be applied to each construct. This derivation is made easier by the constraint that the decomposition occurs one operator at a time in zipper rules. If we allowed for more complex patterns such as $(\lambda x.t) \mathbin{@} s$, we would have to create a switching step for the terms not fitting this pattern, like $x \mathbin{@} s$, and enumerating these anti-patterns would be more difficult [26].

Finally, because we store the annotations of a term in its root operator, we need to remember them when a forward step removes the operator, to be able to restore them when we backtrack. We do so in the stack π.

$$\begin{aligned}\langle t \mathbin{@^{\Sigma}} s\,;\pi \,|\, \mathbb{E}\rangle_{\mathsf{app}} &\mapsto \langle t\,; (\mathsf{appL},\Sigma) :: \pi \,|\, @\, s :: \mathbb{E}\rangle_{\mathsf{app}}\\ \langle (\mathsf{appL},\Sigma) :: \pi\,; t \,|\, @\, s :: \mathbb{E}\rangle_{\mathsf{bapp}} &\mapsto \langle t \mathbin{@^{\Sigma}} s\,;\pi \,|\, \mathbb{E}\rangle_{\mathsf{app}}\end{aligned}$$

In this simple example we could do without the stack because the contexts encode precisely the rules that have been applied along the way. In general, however, a single context cannot always reflect the derivation tree, as we can see in the HOπ example (Appendix C).

The next example illustrates how annotations work, and also that they may no longer hold after reduction. Therefore they should be erased before searching for the next redex.

▸ **Example 2.** Let $\Omega \triangleq (\lambda^{\varnothing} x.x^{\varnothing} \mathbin{@^{\varnothing}} x^{\varnothing}) \mathbin{@^{\varnothing}} (\lambda^{\varnothing} x.x^{\varnothing} \mathbin{@^{\varnothing}} x^{\varnothing})$. We show a possible machine run for this term, where we label forward and backward steps with the rule they apply or unapply, and switching steps with a constant τ. For readability, we write only the term under focus.

The machine may first go left and under the λ-abstraction.

$$\langle \Omega \,|\, \ldots \rangle_{\mathsf{app}} \xmapsto{\mathsf{appL}} \xmapsto{\mathsf{app\lambda}} \langle x^{\varnothing} \mathbin{@^{\varnothing}} x^{\varnothing} \,|\, \ldots \rangle_{\mathsf{app}}$$

At that point, it may test whether the application is a β-redex. Since it is not the case, it backtracks, annotating the variable in function position.

$$\langle x^{\varnothing} \mathbin{@^{\varnothing}} x^{\varnothing} \,|\, \ldots \rangle_{\mathsf{app}} \xmapsto{\mathsf{app\beta}} \langle x^{\varnothing} \,|\, \ldots \rangle_{\mathsf{lam}} \xmapsto{\tau} \xmapsto{-\mathsf{app\beta}} \langle x^{\mathsf{lam}} \mathbin{@^{\varnothing}} x^{\varnothing} \,|\, \ldots \rangle_{\mathsf{app}}$$

From there, it necessarily tests the other possibilities appL and appR (in no predefined order), and fails in both cases.

$$\langle x^{\mathsf{lam}} \mathbin{@^{\varnothing}} x^{\varnothing} \,|\, \ldots \rangle_{\mathsf{app}} \xmapsto{\mathsf{appL}} \xmapsto{\tau} \xmapsto{-\mathsf{appL}} \xmapsto{\mathsf{appR}} \xmapsto{\tau} \xmapsto{-\mathsf{appR}} \langle x^{\{\mathsf{app},\mathsf{lam}\}} @^{\varnothing} x^{\mathsf{app}} \,|\, \ldots \rangle_{\mathsf{app}}$$

Then it can only backtrack to reconstruct the λ-abstraction on the left, and then the whole term.

$$\begin{aligned} \langle x^{\{\mathsf{app},\mathsf{lam}\}} @^{\varnothing} x^{\mathsf{app}} \,|\, \ldots \rangle_{\mathsf{app}} &\xmapsto{\tau} \xmapsto{-\mathsf{app\lambda}} \langle \lambda^{\varnothing} x.x^{\{\mathsf{app},\mathsf{lam}\}} @^{\mathsf{app}} x^{\mathsf{app}} \,|\, \ldots \rangle_{\mathsf{app}} \\ &\xmapsto{\tau} \xmapsto{-\mathsf{appL}} \langle (\lambda^{\mathsf{app}} x.x^{\{\mathsf{app},\mathsf{lam}\}} @^{\mathsf{app}} x^{\mathsf{app}}) \mathbin{@^{\varnothing}} (\lambda^{\varnothing} x.x^{\varnothing} \mathbin{@^{\varnothing}} x^{\varnothing}) \,|\, \ldots \rangle_{\mathsf{app}} \end{aligned}$$

The machine can then look for a redex in the λ-abstraction on the right, and it would result in the same annotations as for the one on the left, not necessarily generated in the same order. It can also rightfully recognize the term as a β-redex, with the sequence $\xmapsto{\mathsf{app\beta}} \xmapsto{\mathsf{lam\beta}}$, the last step performing the reduction. After the reduction, we should also erase the remaining annotations. If we do not erase them, the result of the reduction would be

$$\langle (\lambda^{\varnothing} x.x^{\varnothing} \mathbin{@^{\varnothing}} x^{\varnothing}) @^{\{\mathsf{app}\}} (\lambda^{\varnothing} x.x^{\varnothing} \mathbin{@^{\varnothing}} x^{\varnothing}) \,|\, \ldots \rangle_{\mathsf{app}}$$

and the app annotation would wrongfully signal the term as a normal-form, preventing it from being reduced. Erasing all the remaining annotations ensures the machine finds the next redex, but a finer, language-specific analysis would erase only the problematic annotations. We leave such an optimization as a future work. ◂

Formal definition. We let α range over annotations, Σ over annotation sets, and denote the empty set by $\varnothing$. We extend the λ-calculus syntax as follows:

$$\alpha ::= \mathsf{app} \mid \mathsf{lam} \qquad t, s ::= x^{\Sigma} \mid \lambda^{\Sigma} x.t \mid t \mathbin{@^{\Sigma}} s$$

We write $\mathsf{an}(t)$ for the annotation set at the root of t, e.g., $\mathsf{an}(t \mathbin{@^{\Sigma}} s) \triangleq \Sigma$. We write $t^{\cup \alpha}$ for its extension with α so that $\mathsf{an}(t^{\cup\alpha}) = \mathsf{an}(t) \cup \{\alpha\}$. We write $|t|$ for the erasure of t, where all the annotation sets in t are made empty.

The syntax of contexts uses annotated terms, and plugging returns an annotated term where the annotation sets of the context operators are empty: e.g., $(\lambda x :: \mathbb{E})[t] \triangleq \mathbb{E}[\lambda^{\varnothing} x.t]$. Plugging is used only after a reduction step, where all the annotation sets are erased anyway.

We let ρ range over rule names and π over rule stacks, defined as $\pi ::= \mathsf{init} \mid (\rho, \Sigma) :: \pi$. The definition of the machine for the λ-calculus is given in Figure 2.

$$
\begin{array}{rcll}
\langle t \rangle_{\mathsf{zs}} & \mapsto & \langle t \,;\, \mathsf{init} \mid \bullet \rangle_{\mathsf{app}} & \\[1ex]
\langle t \mathbin{@^{\Sigma}} s \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{app}} & \mapsto & \langle t \,;\, (\mathsf{appL}, \Sigma) :: \pi \mid @\, s :: \mathbb{E} \rangle_{\mathsf{app}} & \text{if } \mathsf{app} \notin \mathsf{an}(t) \\
\langle t \mathbin{@^{\Sigma}} s \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{app}} & \mapsto & \langle s \,;\, (\mathsf{appR}, \Sigma) :: \pi \mid t\, @ :: \mathbb{E} \rangle_{\mathsf{app}} & \text{if } \mathsf{app} \notin \mathsf{an}(s) \\
\langle t \mathbin{@^{\Sigma}} s \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{app}} & \mapsto & \langle t \,;\, (\mathsf{app\beta}, \Sigma) :: \pi \mid s, \mathbb{E} \rangle_{\mathsf{lam}} & \text{if } \mathsf{lam} \notin \mathsf{an}(t) \\
\langle \lambda^{\Sigma} x.t \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{app}} & \mapsto & \langle t \,;\, (\mathsf{app\lambda}, \Sigma) :: \pi \mid \lambda x :: \mathbb{E} \rangle_{\mathsf{app}} & \text{if } \mathsf{app} \notin \mathsf{an}(t) \\
\langle t \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{app}} & \mapsto & \langle \pi \,;\, t^{\cup \mathsf{app}} \mid \mathbb{E} \rangle_{\mathsf{bapp}} & \text{otherwise} \\[1ex]
\langle \mathsf{init} \,;\, t \mid \bullet \rangle_{\mathsf{bapp}} & \mapsto & \langle t \rangle_{\mathsf{nf}} & \\
\langle (\mathsf{appL}, \Sigma) :: \pi \,;\, t \mid @\, s :: \mathbb{E} \rangle_{\mathsf{bapp}} & \mapsto & \langle t \mathbin{@^{\Sigma}} s \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{app}} & \\
\langle (\mathsf{appR}, \Sigma) :: \pi \,;\, s \mid t\, @ :: \mathbb{E} \rangle_{\mathsf{bapp}} & \mapsto & \langle t \mathbin{@^{\Sigma}} s \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{app}} & \\
\langle (\mathsf{app\beta}, \Sigma) :: \pi \,;\, t \mid s, \mathbb{E} \rangle_{\mathsf{blam}} & \mapsto & \langle t \mathbin{@^{\Sigma}} s \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{app}} & \\
\langle (\mathsf{app\lambda}, \Sigma) :: \pi \,;\, t \mid \lambda x :: \mathbb{E} \rangle_{\mathsf{bapp}} & \mapsto & \langle \lambda^{\Sigma} x.t \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{app}} & \\[1ex]
\langle \lambda^{\Sigma} x.t \,;\, \pi \mid s, \mathbb{E} \rangle_{\mathsf{lam}} & \mapsto & \langle |\mathbb{E}[t\{s/x\}]| \rangle_{\mathsf{zs}} & \\
\langle t \,;\, \pi \mid s, \mathbb{E} \rangle_{\mathsf{lam}} & \mapsto & \langle \pi \,;\, t^{\cup \mathsf{lam}} \mid s, \mathbb{E} \rangle_{\mathsf{blam}} & \text{otherwise}
\end{array}
$$

Figure 2 Non-Deterministic Abstract Machine for the λ-calculus

A *forward* configuration $\langle t \,;\, \pi \mid \mathbb{E} \rangle_{\mathsf{m}}$ (with $\mathsf{m} \in \{\mathsf{app}, \mathsf{lam}\}$) discriminates on (the root operator of) t to apply a rule of the zipper semantics. For an inductive rule, it results in a change of focus and an extension of the stack, on which we record the applied rule and the annotation set of the root operator. Taking such a step is possible only if the new term under focus is not a normal form. A special case of forward step is the *initial* one from $\langle t \rangle_{\mathsf{zs}}$ which does not have a side-condition, as we assume the annotation sets of t to be empty.

The β-reduction happens in the first transition of the lam mode. Backtracking is no longer necessary so we drop the stack. We reconstruct the entire term, and switch to the initial mode to search for a new redex starting from the root of the new term. We erase all annotations, as they may no longer be valid, as illustrated by Example 2.

If a forward configuration cannot apply a rule, we switch to the corresponding *backward* mode, annotating t in the process: these are the two "otherwise" steps. A backward configuration $\langle \pi \,;\, t \mid \mathbb{E} \rangle_{\mathsf{bm}}$ inspects the stack π to unapply the rule at its top. While a backward step restores the configuration of the corresponding forward step, the term contains more annotations after a backward step than before taking the forward step: in $\langle \pi \,;\, t \mid \mathbb{E} \rangle_{\mathsf{bm}}$, we have $\mathsf{m} \in \mathsf{an}(t)$ by construction. The annotations prevent the machine from reapplying a rule it just unapplied. The *normal form* mode $\langle t \rangle_{\mathsf{nf}}$ signals that the term cannot reduce.

A machine run starts with an initial configuration $\langle t \rangle_{\mathsf{zs}}$ where all the annotation sets of t are empty. The semantics of the machine is given by these configurations: if $\langle t \rangle_{\mathsf{zs}} \mapsto^{+} \langle t' \rangle_{\mathsf{zs}}$ such that the sequence $\mapsto^{+}$ does not go through another initial configuration, then $t \to_{\mathsf{zs}} t'$. Similarly, if $\langle t \rangle_{\mathsf{zs}} \mapsto^{+} \langle t' \rangle_{\mathsf{nf}}$, then $|t'| = t$ and t is a normal form. We state the correspondence and termination theorems independently from the source zipper semantics in Section 4.

$$\textsf{init}\ \frac{P \xrightarrow{\bullet}_{\mathsf{par}} P'}{P \to_{\mathsf{zs}} P'} \qquad \textsf{parL}\ \frac{P \xrightarrow{\parallel Q :: \mathbb{E}}_{\mathsf{par}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'}\,(s) \qquad \textsf{parOutL}\ \frac{P \xrightarrow{\bullet, \mathcal{L}, \mathbb{E}, Q}_{\mathsf{out}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'} \qquad \textsf{parOutR}\ \frac{Q \xrightarrow{\bullet, \mathcal{R}, \mathbb{E}, P}_{\mathsf{out}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'}$$

$$\textsf{outParL}\ \frac{P \xrightarrow{\parallel Q :: \mathbb{F}, \mathcal{S}, \mathbb{E}, R}_{\mathsf{out}} P'}{P \parallel Q \xrightarrow{\mathbb{F}, \mathcal{S}, \mathbb{E}, R}_{\mathsf{out}} P'}\,(s) \qquad \textsf{outIn}\ \frac{R \xrightarrow{\bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}}_{\mathsf{in}} P'}{\overline{a}\langle P \rangle \xrightarrow{\mathbb{F}, \mathcal{S}, \mathbb{E}, R}_{\mathsf{out}} P'} \qquad \textsf{inParL}\ \frac{R \xrightarrow{\parallel Q :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}}_{\mathsf{in}} P'}{R \parallel Q \xrightarrow{\mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}}_{\mathsf{in}} P'}\,(s)$$

$$\textsf{inComL}\ \frac{b = a}{b(X).R \xrightarrow{\mathbb{G}, \mathcal{L}, a, P, \mathbb{E}, \mathbb{F}}_{\mathsf{in}} \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]}\,(s)$$

Figure 3 Output-first Zipper Semantics for HOcore

3 HOcore

We consider a minimal process calculus called HOcore [30], which can be seen as an extension of the λ-calculus with parallel composition.

3.1 Syntax and Semantics

We let a, b range over *channel names*, X, Y over *process variables*, and we define the syntax of processes as follows.

$$P, Q, R ::= X \mid \mathbf{0} \mid P \parallel Q \mid a(X).P \mid \overline{a}\langle P \rangle$$

The process $\mathbf{0}$ is the inactive process, $P \parallel Q$ runs P and Q in parallel, and a communication may happen between an input $a(X).P$ and an output $\overline{a}\langle Q \rangle$ that run in parallel. The communication is asynchronous because a message output does not have a continuation [42]; we discuss the synchronous case in Remark 3. In spite of its minimal number of constructors, HOcore is Turing-complete [30].

The semantics of process calculi is usually presented either with a structural congruence relation which reorders terms to make redexes appear, bringing input and output processes together, or with a labeled transition system which preserves the structure of the term [42]. Instead, we present it first as a reduction semantics with explicit contexts, as in Section 2.1, which makes it easier to come up with (or translate into) the corresponding zipper semantics.

We define frames as $\mathfrak{F} ::= \parallel P \mid P \parallel$ and plugging as follows.

$$\bullet[P] \triangleq P \qquad (\parallel Q :: \mathbb{E})[P] \triangleq \mathbb{E}[P \parallel Q] \qquad (Q \parallel :: \mathbb{E})[P] \triangleq \mathbb{E}[Q \parallel P]$$

A redex is a parallel composition with an input on one side and an output on the same name on the other side, both surrounded with contexts. The general formulation of such communication sites in a program can be expressed with the following reduction semantics, where we write $P\{Q/X\}$ for the capture-avoiding substitution of X by Q in P:

$$\mathbb{E}[\mathbb{F}[\overline{a}\langle Q \rangle] \parallel \mathbb{G}[a(X).P]] \to_{\mathsf{rs}} \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[P\{Q/X\}]]$$
$$\mathbb{E}[\mathbb{G}[a(X).P] \parallel \mathbb{F}[\overline{a}\langle Q \rangle]] \to_{\mathsf{rs}} \mathbb{E}[\mathbb{G}[P\{Q/X\}] \parallel \mathbb{F}[\mathbf{0}]]$$

3.2 Zipper Semantics

Finding an HOcore redex requires us to recognize three constructs (parallel composition along with output and input on a shared name) and build the contexts $\mathbb{E}$, $\mathbb{F}$, and $\mathbb{G}$. The first step is to find the parallel composition; once the communicating processes $P \parallel Q$ are found, the communication rules of typical LTSs for process calculi [30, 41, 42] have two premises looking for the output and the input in P and Q respectively. To be closer to an abstract machine, we sequentialize the search by looking for the output first (while constructing $\mathbb{F}$) and then the input (with $\mathbb{G}$)—the opposite choice would produce a completely symmetric semantics. Figure 3 presents such an output-first zipper semantics, where we omit the symmetric versions of the rules marked with the symbol (s). The resulting semantics is close to complementary semantics [31], where the communication is also sequentialized.

The transition $\xrightarrow{\mathbb{E}}_{\mathsf{par}}$ is looking for the parallel composition while building $\mathbb{E}$: it proceeds as $\xrightarrow{\mathbb{E}}_{\mathsf{app}}$ in the λ-calculus. Once we find the parallel composition, we look for the output either on the left or on the right with respectively rules parOutL and parOutR. We record the side we pick with a parameter $\mathcal{S} ::= \mathcal{L} \mid \mathcal{R}$. For example, in rule parOutL, we look for an output in P on the left ($\mathcal{L}$), remembering that we should later search for a corresponding input in Q. We also initialize the context $\mathbb{F}$ surrounding the output with $\bullet$ and remember $\mathbb{E}$ as the context enclosing the whole redex.

The transition $\xrightarrow{\mathbb{F},\mathcal{S},\mathbb{E},R}_{\mathsf{out}}$ decomposes its source process to find an output, building $\mathbb{F}$ at the same time: the other parameters $\mathcal{S}$, $\mathbb{E}$, and R remain unchanged during the search. When we find the output $\overline{a}\langle P\rangle$ (rule outIn), we look for a corresponding input in R using $\xrightarrow{\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}}_{\mathsf{in}}$, which builds the context $\mathbb{G}$ during the search. Once we find an input on a, we compute the result of the communication, which depends whether the output is on the left (rule inComL) or on the right (omitted rule inComR).

We prove the correspondence between the two semantics in Appendix B.

▸ Remark 3 (Synchronous communication). For a synchronous calculus with an output $\overline{a}\langle P\rangle Q$, the rule outIn would pass the continuation Q as an argument of the input transition in. The continuation Q would then be plugged into $\mathbb{F}$ in the axioms inComL and inComR.

▸ Remark 4 (Left-first search). After finding the communicating processes $P \parallel Q$, we could always go left (in P). When we find an output or input in P, we look for its complement in Q. A right-first search is also possible. We present the left-first zipper semantics and its machine in Appendix B; such an approach does not scale to HOπ, as explained in Remark 22.

3.3 Non-Deterministic Abstract Machine

We derive the HOcore NDAM from its zipper semantics along the same principles as for the λ-calculus: each rule of the semantics corresponds to a forward step and a backward step, and when no forward step applies to a configuration, we switch to a backward configuration. The difference is in the normal-form annotations: in λ-calculus, to be a normal form w.r.t. $\xrightarrow{s,\mathbb{E}}_{\mathsf{lam}}$ or $\xrightarrow{\mathbb{E}}_{\mathsf{app}}$ does not depend on the arguments s and $\mathbb{E}$. In HOcore, being a normal form depends on some of the arguments in the input and output transitions.

For example, in a process $(\overline{a}\langle\mathbf{0}\rangle \parallel \overline{b}\langle\mathbf{0}\rangle) \parallel Q$, we may look into Q for an input on a or on b. If Q does not contain an input on a, then annotating it with the mode in would prevent from searching in Q for an input on b. We therefore include the name in the annotation, marking the root operator of Q with (in, a), meaning that Q cannot do an input on a. If it also cannot do an input on b, then its root operator will be annotated with both (in, a) and (in, b).

$$\langle P\rangle_{\mathsf{zs}} \mapsto \langle P \mathbin{;} \mathsf{init} \mid \bullet\rangle_{\mathsf{par}}$$

$$\begin{array}{rcll}
\langle P \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{E}\rangle_{\mathsf{par}} & \mapsto & \langle P \mathbin{;} (\mathsf{parL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{E}\rangle_{\mathsf{par}} & \text{if } \mathsf{par} \notin \mathsf{an}(P) \\
\langle P \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{E}\rangle_{\mathsf{par}} & \mapsto & \langle P \mathbin{;} (\mathsf{parOutL}, \Sigma) :: \pi \mid \bullet, \mathcal{L}, \mathbb{E}, Q\rangle_{\mathsf{out}} & \text{if } (\mathsf{out}, |Q|) \notin \mathsf{an}(P) \\
\langle P \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{E}\rangle_{\mathsf{par}} & \mapsto & \langle Q \mathbin{;} (\mathsf{parOutR}, \Sigma) :: \pi \mid \bullet, \mathcal{R}, \mathbb{E}, P\rangle_{\mathsf{out}} & \text{if } (\mathsf{out}, |P|) \notin \mathsf{an}(Q) \\
\langle P \mathbin{;} \pi \mid \mathbb{E}\rangle_{\mathsf{par}} & \mapsto & \langle \pi \mathbin{;} P^{\cup \mathsf{par}} \mid \mathbb{E}\rangle_{\mathsf{bpar}} & \text{otherwise}
\end{array}$$

$$\begin{array}{rcl}
\langle \mathsf{init} \mathbin{;} P \mid \bullet\rangle_{\mathsf{bpar}} & \mapsto & \langle P\rangle_{\mathsf{nf}} \\
\langle (\mathsf{parL}, \Sigma) :: \pi \mathbin{;} P \mid \parallel Q :: \mathbb{E}\rangle_{\mathsf{bpar}} & \mapsto & \langle P \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{E}\rangle_{\mathsf{par}}
\end{array}$$

$$\begin{array}{rcll}
\langle P \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R\rangle_{\mathsf{out}} & \mapsto & \langle P \mathbin{;} (\mathsf{outParL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{F}, \mathcal{S}, \mathbb{E}, R\rangle_{\mathsf{out}} & \text{if } (\mathsf{out}, |R|) \notin \mathsf{an}(P) \\
\langle \overline{a}^{\Sigma}\langle P\rangle \mathbin{;} \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R\rangle_{\mathsf{out}} & \mapsto & \langle R \mathbin{;} (\mathsf{outIn}, \Sigma) :: \pi \mid \bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \text{if } (\mathsf{in}, a) \notin \mathsf{an}(R) \\
\langle P \mathbin{;} \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R\rangle_{\mathsf{out}} & \mapsto & \langle \pi \mathbin{;} P^{\cup (\mathsf{out}, |R|)} \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R\rangle_{\mathsf{bout}} & \text{otherwise}
\end{array}$$

$$\begin{array}{rcl}
\langle (\mathsf{parOutL}, \Sigma) :: \pi \mathbin{;} P \mid \bullet, \mathcal{L}, \mathbb{E}, Q\rangle_{\mathsf{bout}} & \mapsto & \langle P \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{E}\rangle_{\mathsf{par}} \\
\langle (\mathsf{parOutR}, \Sigma) :: \pi \mathbin{;} Q \mid \bullet, \mathcal{R}, \mathbb{E}, P\rangle_{\mathsf{bout}} & \mapsto & \langle P \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{E}\rangle_{\mathsf{par}} \\
\langle (\mathsf{outParL}, \Sigma) :: \pi \mathbin{;} P \mid \parallel Q :: \mathbb{F}, \mathcal{S}, \mathbb{E}, R\rangle_{\mathsf{bout}} & \mapsto & \langle P \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R\rangle_{\mathsf{out}}
\end{array}$$

$$\begin{array}{rcll}
\langle R \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \mapsto & \langle R \mathbin{;} (\mathsf{inParL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \text{if } (\mathsf{in}, a) \notin \mathsf{an}(R) \\
\langle b^{\Sigma}(X).R \mathbin{;} \pi \mid \mathbb{G}, \mathcal{L}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \mapsto & \langle |\mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]|\rangle_{\mathsf{zs}} & \text{if } a = b \\
\langle b^{\Sigma}(X).R \mathbin{;} \pi \mid \mathbb{G}, \mathcal{R}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \mapsto & \langle |\mathbb{E}[\mathbb{G}[R\{P/X\}] \parallel \mathbb{F}[\mathbf{0}]]|\rangle_{\mathsf{zs}} & \text{if } a = b \\
\langle R \mathbin{;} \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \mapsto & \langle \pi \mathbin{;} R^{\cup (\mathsf{in}, a)} \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{bin}} & \text{otherwise}
\end{array}$$

$$\begin{array}{rcl}
\langle (\mathsf{outIn}, \Sigma) :: \pi \mathbin{;} R \mid \bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{bin}} & \mapsto & \langle \overline{a}^{\Sigma}\langle P\rangle \mathbin{;} \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R\rangle_{\mathsf{out}} \\
\langle (\mathsf{inParL}, \Sigma) :: \pi \mathbin{;} R \mid \parallel Q :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{bin}} & \mapsto & \langle R \parallel^{\Sigma} Q \mathbin{;} \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}}
\end{array}$$

Figure 4 Non-Deterministic Abstract Machine for HOcore

With outputs the problem is similar, but not completely symmetric. Let $P_{a,b} = \overline{a}\langle \mathbf{0}\rangle \parallel \overline{b}\langle \mathbf{0}\rangle$, and consider a process $(P_{a,b} \parallel Q) \parallel R$. We may try to find a communication between $P_{a,b}$ and Q first. If Q does not contain an input on a or b, then $P_{a,b}$ is a normal form w.r.t. the output search transition $\xrightarrow{\bullet, \mathcal{L}, \parallel R :: \bullet, Q}_{\mathsf{out}}$, but a communication between $P_{a,b}$ and R is still possible. As a result, we annotate the root operator of $P_{a,b}$ with (out, Q), meaning that the outputs of $P_{a,b}$ are not complemented by the inputs in Q. Such an annotation does not prevent trying to make $P_{a,b}$ and R communicate, which would correspond to the transition $\xrightarrow{\parallel Q :: \bullet, \mathcal{L}, \bullet, R}_{\mathsf{out}}$.

As before, Σ ranges over annotation sets, and $|P|$ is the erasure of P, the annotated process with empty annotation sets. The syntax of annotations and processes is as follows.

$$\alpha ::= \mathsf{par} \mid (\mathsf{out}, |P|) \mid (\mathsf{in}, a) \qquad P, Q, R ::= X^{\Sigma} \mid \mathbf{0}^{\Sigma} \mid P \parallel^{\Sigma} Q \mid a^{\Sigma}(X).P \mid \overline{a}^{\Sigma}\langle P\rangle$$

Substitution and plugging are extended to annotated processes as expected. The definition of the machine is given in Figure 4. The process P in an annotation $(\mathsf{out}, |P|)$—as in the side conditions in the par-transitions—is erased, because normal forms are defined with respect to the zipper semantics transitions, where processes are not annotated. Apart from richer annotations, the definition of the machine follows the principles of Section 2.3. Note that the "otherwise" step for the input mode includes the operators that are not parsed in that mode, but also the inputs on a name distinct from a.

4 Derivation of the Abstract Machine

We show how to derive an abstract machine from a zipper semantics under some conditions. To this end, we specify zipper semantics as a transition system [21], a framework used to describe rule formats.

4.1 Zipper Semantics as a Transition System

Given an entity e, we write $\widetilde{e}$ for a possibly empty sequence $(e_1, \ldots, e_n)$ for some n. We assume a set $\mathcal{S}$ of sorts ranged over by s, denoting the entities of the language (contexts, names, etc), and which includes the sort t of terms that are reduced. For each sort s, let $\mathcal{O}_s$ be the signature of s, i.e., a set of operators, each having a typing $\widetilde{s} \to s$. In particular, we let op range over the operators of the terms $\mathcal{O}_t$. We also assume a set $\mathcal{F}$ of auxiliary functions that are used to build terms, like term substitution or context plugging, each of type $\widetilde{s} \to t$.

For each s, we assume an infinite set $\mathcal{V}_s$ of *rule variables*, denoted by v_s, w_s, or v, w if the sort does not matter. The set $\mathfrak{E}_s$ of *rule entities* of sort s, ranged over by e_s, f_s (or e, f if we ignore the sort), are the entities built out of the signature $\mathcal{O}_s$ extended with rule variables. We define $\mathfrak{E}_s$ inductively so that $\mathcal{V}_s \subseteq \mathfrak{E}_s$, and for all $o \in \mathcal{O}_s$ of signature $(s_1, \ldots, s_n) \to s$ and $(e_{s_i} \in \mathfrak{E}_{s_i})_{i \in 1 \ldots n}$ for some n, we have $o(e_{s_1}, \ldots, e_{s_n}) \in \mathfrak{E}_s$. A special case are term entities e_t, which can also be built out of auxiliary functions in $\mathcal{F}$. We write $\mathsf{rv}(e_s)$ for the set of rule variables of e_s; e_s is ground if $\mathsf{rv}(e_s) = \varnothing$.

A rule substitution σ is a sort-respecting mapping from rule variables to rule entities. It should not be confused with the substitution $\cdot\{\cdot/\cdot\}$ which may exist for terms and is considered an auxiliary function in $\mathcal{F}$. We write $v\sigma$ for the application of σ to v, and $e\sigma$—for its extension to rule entities, defined in the expected way. A ground entity e is an instance of e' if there exists σ such that $e'\sigma = e$.

Given some rule variables $\widetilde{v}$, we write $\mathcal{P}(\widetilde{v})$ for a decidable predicate on $\widetilde{v}$. We assume a set $\mathcal{M}$ of modes, denoted by m, such that each mode is associated with a sequence $\widetilde{s_{\mathsf{m}}}$ giving the sorts of its arguments. The set $\mathcal{M}$ includes the initial mode zs with no argument.

A *transition* is a predicate $e_i \xrightarrow{\widetilde{e}}_{\mathsf{m}} e_o$, where e_i and e_o are respectively the source and the target. We consider only three kinds of rule: inductive (whose names are ranged over with ρ), axiom, and initial, of the following respective shapes.

$$\frac{e_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'} v_t \qquad \mathcal{P}(\widetilde{w})}{op(\widetilde{v}) \xrightarrow{\widetilde{e}}_{\mathsf{m}} v_t}\ \rho \qquad\qquad \frac{\mathcal{P}(\widetilde{w})}{op(\widetilde{v}) \xrightarrow{\widetilde{e}}_{\mathsf{m}} e_t} \qquad\qquad \frac{v_t \xrightarrow{\widetilde{f}}_{\mathsf{m}} w_t}{v_t \to_{\mathsf{zs}} w_t}\ \mathsf{init}$$

We extend the notion of set of rule variables rv and the application of a substitution to transitions and rules.

An inductive rule has only one premise, and may have side-conditions, represented by $\mathcal{P}$, on some of the variables $\widetilde{w}$ occurring in the rule. The modes m and m' may be distinct or not, and the sequences $\widetilde{e}$ and $\widetilde{f}$ should be rule entities of sorts respectively $\widetilde{s_{\mathsf{m}}}$ and $\widetilde{s_{\mathsf{m}'}}$. The sources and targets of the transitions are terms; in the conclusion, the source term is of the form $op(\widetilde{v})$, enforcing that a rule can only pattern-match the head operator of the term. Both targets should be the same term variable, meaning that an inductive rule is simply passing along the result. Computation occurs in axioms, where the target can be any term.

$$\textsf{toy}\ \frac{P \xrightarrow{a,\mathbb{E}} P'}{P \xrightarrow{\mathbb{E}} P'} \qquad \textsf{outInL}\ \frac{R \xrightarrow{\mathbb{F}[\mathbf{0}] \parallel :: \mathbb{E}, a, P}_{\mathsf{in}} P'}{\overline{a}\langle P\rangle \xrightarrow{\mathbb{F},\mathcal{L},\mathbb{E},R}_{\mathsf{out}} P'} \qquad \textsf{choiceBad}\ \frac{P \xrightarrow{\mathbb{E}} P'}{P + Q \xrightarrow{\mathbb{E}} P'} \qquad \textsf{choiceOk}\ \frac{P \xrightarrow{\mathbb{E}, Q :: \theta} P'}{P + Q \xrightarrow{\mathbb{E},\theta} P'} \qquad \textsf{rec}\ \frac{P\{\mu X.P/X\} \xrightarrow{\mathbb{E}} P'}{\mu X.P \xrightarrow{\mathbb{E}} P'}$$

Figure 5 Rules for variants of HOcore

An initial rule defines the initial mode zs. The source of the conclusion is a variable, so an initial rule does not perform any pattern-matching. An initial rule is just a means to set up the arguments of another mode m (such that $\mathsf{m} \neq \mathsf{zs}$). A zipper semantics is a triple $(\mathcal{S}, \mathcal{O}, \mathcal{R})$ where $\mathcal{R}$ is a finite set of zipper rules with exactly one initial rule. The associated semantics on terms is defined by $\rightarrow_{\mathsf{zs}}$.

4.2 Derivable Zipper Semantics

Not every zipper semantics can be turned into an NDAM. Some conditions have to be satisfied for the transformation to be possible and to ensure termination.

The first one is that the rules of the semantics must be constructive w.r.t. the machine, meaning that the entities in its premise are constructed from the ones in the conclusion. Indeed, the abstract machine searches for redexes with forward steps by going from the conclusion to the premise of a rule. As a result, a rule like toy in Figure 5 cannot be turned into a machine step, as the machine would have to guess the name a. We forbid such a rule by requiring that in each inductive rule of the zipper semantics, the rule variables of the premise are included in the rule variables of the conclusion.

▸ **Definition 5.** $\dfrac{e_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'} v_t \quad \mathcal{P}(\widetilde{w})}{op(\widetilde{v}) \xrightarrow{\widetilde{e}}_{\mathsf{m}} v_t}$ *is machine constructive if* $\mathsf{rv}(e_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'}) \cup \widetilde{w} \subseteq \mathsf{rv}(op(\widetilde{v}) \xrightarrow{\widetilde{e}}_{\mathsf{m}})$.

The other constraint is that the rules must be *reversible* to allow for backtracking: it should be possible to reconstruct the entities in the conclusion from the ones in the premise. We say a rule is reversible if it cannot have two different instances with the same premise. For example, we could make the input search in HOcore less verbose, by combining the contexts $\mathbb{E}$ and $\mathbb{F}$ in a single context, like in the rule outInL in Figure 5. In $\mathbb{E}[R \parallel \mathbb{F}[\mathbf{0}]]$, the input process is plugged into the context $\parallel \mathbb{F}[\mathbf{0}] :: \mathbb{E}$, that we build in rule outInL, instead of keeping $\mathbb{E}$ and $\mathbb{F}$ separate as in Figure 3. However, to unapply the rule outInL, we need to uniquely decompose a context as $\parallel \mathbb{F}[\mathbf{0}] :: \mathbb{E}$, which is not possible as soon as there are several occurrences of $\mathbf{0}$ in $\mathbb{F}[\mathbf{0}]$: the rule outInL is not reversible. We give a simple sufficient criterion for a rule to be reversible.

▸ **Lemma 6.** $\dfrac{e_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'} v_t \qquad \mathcal{P}(\widetilde{w})}{op(\widetilde{v}) \xrightarrow{\widetilde{e}}_{\mathsf{m}} v_t}$ *is reversible if we have* $\mathsf{rv}(op(\widetilde{v}) \xrightarrow{\widetilde{e}}_{\mathsf{m}}) \subseteq \mathsf{rv}(e_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'})$*, and the auxiliary functions used to build the entities in* e_t *and* $\widetilde{f}$ *are injective.*

The first condition states that the rules variables of the conclusion have to be included in those of the premise. Indeed, if we forget an entity between the conclusion and the premise, like Q in the rule for choice $\mathsf{choiceBad}$ in Figure 5, then we have no information to restore Q when backtracking. Instead, it should be kept in an extra argument of the zipper semantics, like the stack θ in the rule $\mathsf{choiceOk}$ in Figure 5. The stack θ is useful only for backtracking and not to define the semantics of the language, as it is simply thrown away when we apply an axiom. Any rule forgetting entities between its conclusion and premise can be made reversible using this principle [38].

Finally, we want the machine to always terminate when searching for a redex. Consider for instance the rec rule for a recursion operator in Figure 5. The corresponding machine would infinitely loop with $\mu X.X$. Indeed, the forward step of this rule changes focus from the source of the conclusion to the source of the premise, but these two terms are equal when $P = X$. To avoid this, we require the zipper semantics to be well-founded.

▸ **Definition 7.** *A zipper semantics is well-founded if there exists a well-founded size ζ such that for all inductive rules $\dfrac{e'_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'} v_t \qquad \mathcal{P}(\widetilde{w})}{e_t \xrightarrow{\widetilde{e}}_{\mathsf{m}} v_t}$, we have $\zeta(e'_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'} v_t) < \zeta(e_t \xrightarrow{\widetilde{e}}_{\mathsf{m}} v_t)$.*

In the calculi of this paper, each rule either focuses on a subterm or it changes mode (like in rule outIn in HOcore). We therefore define an ordering on modes such that $\mathsf{m} > \mathsf{m}'$ if the derivation of m depends on m'; e.g., we have $\mathsf{zs} > \mathsf{app} > \mathsf{lam}$ in λ-calculus, and $\mathsf{zs} > \mathsf{par} > \mathsf{out} > \mathsf{in}$ in HOcore and HOπ. The size we consider is then the lexicographic ordering composed of the ordering on modes followed by the subterm ordering on the source term of the transition. This size works as long as we have no cyclic dependencies in modes and only congruence rules within each mode. It rules out unconstrained recursion, but we can still adapt it for guarded recursion, where the recursion variable occurs only after an input, as in $\mu X.a(Y).(X \parallel Y)$. In the premise of the rec rule, the μ operator itself becomes guarded, so the number of recursion operators at toplevel strictly decreases.

The semantics of Figures 1, 3, and 9 are machine constructive well-founded, and reversible (they satisfy Lemma 6). Henceforth, we assume the zipper semantics to be machine constructive, reversible, and well-founded.

4.3 Machine Derivation

Annotations. The machine annotates terms which cannot do certain transitions, to forbid repeated tries which would lead to an infinite loop. The arguments of the transition may play a role in whether the term is a normal form or not: in HOcore an output $\overline{a}\langle P\rangle$ is a normal form w.r.t. the output transition $\xrightarrow{\mathbb{F},\mathcal{S},\mathbb{E},R}_{\mathsf{out}}$ if R cannot receive the message on a, so the annotation is $(\mathsf{out}, |R|)$. Similarly an input $\xrightarrow{\mathbb{G},\mathcal{S},a,\mathbb{E},\mathbb{F}}_{\mathsf{in}}$ depends on the name a.

The arguments kept in the annotation are the ones either taking part in the reduction, like R in the output case, or in side-conditions, like a in the input case. Given a mode m with arguments $\widetilde{e}$, its annotation $\phi(\mathsf{m}, \widetilde{e})$ is defined as $(\mathsf{m}, \widetilde{f})$ where $\widetilde{f} \subseteq \widetilde{e}$ are the arguments occurring either in side-conditions or source terms of the rules defining m. Repeating this for each mode of a zipper semantics, we define the *annotation function* ϕ of the semantics.

Annotated terms. Let $(\mathcal{S}, \mathcal{O}, \mathcal{R})$ be a zipper semantics with annotation function ϕ. We extend $\mathcal{S}$ with the sort of annotation sets s_Σ, for which we assume the usual operators on sets. The machine is built on a signature $\mathcal{A}$ which replaces the signature for terms $\mathcal{O}_t$ with *annotated terms*, so that for all $op \in \mathcal{O}_t$ of type $(s_1, \ldots, s_n) \to t$ for some n, we have a corresponding operator $op \in \mathcal{A}_t$ of type $(s_\Sigma, s_1, \ldots, s_n) \to t$.

We let a range over annotated terms $\mathfrak{A}_t$, built out of $\mathcal{A}$, $\mathcal{V}_t$, and a single rule variable for annotation sets v_Σ: one variable is enough, as at most one annotation set occurs in a given machine step. Given an annotated term $a = op(e_\Sigma, \widetilde{e})$, we write $\mathsf{an}(a)$ for its annotation set e_Σ. Given a term $e_t \in \mathfrak{E}_t$, its annotated version, written $\|e_t\|$, is inductively defined so

$$\dfrac{e_t \xrightarrow{\tilde{f}}_{\mathsf{m}'} v_t \qquad \mathcal{P}(\tilde{w})}{op(\tilde{v}) \xrightarrow{\tilde{e}}_{\mathsf{m}} v_t}\,\rho \qquad \begin{array}{rl} \langle op(v_\Sigma, \tilde{v})\,;\pi \mid \tilde{e}\rangle_{\mathsf{m}} & \mapsto \langle \|e_t\|\,;(\rho, v_\Sigma) :: \pi \mid \tilde{f}\rangle_{\mathsf{m}'} \\ & \quad \text{if } \phi(\mathsf{m}', \tilde{f}) \notin \mathsf{an}(\|e_t\|) \text{ and } \mathcal{P}(\tilde{w}) \\ \langle (\rho, v_\Sigma) :: \pi\,; \|e_t\| \mid \tilde{f}\rangle_{\mathsf{bm}'} & \mapsto \langle op(v_\Sigma, \tilde{v})\,;\pi \mid \tilde{e}\rangle_{\mathsf{m}} \end{array}$$

$$\dfrac{v_t \xrightarrow{\tilde{f}}_{\mathsf{m}} w_t}{v_t \rightarrow_{\mathsf{zs}} w_t}\,\mathsf{init} \qquad \begin{array}{rl} \langle v_t\rangle_{\mathsf{zs}} & \mapsto \langle v_t\,;\mathsf{init} \mid \tilde{f}\rangle_{\mathsf{m}} \\ \langle \mathsf{init}\,; v_t \mid \tilde{f}\rangle_{\mathsf{bm}} & \mapsto \langle v_t\rangle_{\mathsf{nf}} \end{array}$$

$$\dfrac{\mathcal{P}(\tilde{w})}{op(\tilde{v}) \xrightarrow{\tilde{e}}_{\mathsf{m}} e_t} \qquad \langle op(v_\Sigma, \tilde{v})\,;\pi \mid \tilde{e}\rangle_{\mathsf{m}} \mapsto \langle |\,\|e_t\|\,|\rangle_{\mathsf{zs}} \text{ if } \mathcal{P}(\tilde{w})$$

Figure 6 Forward and backward steps generated from a zipper semantics rule

that $\|v_s\| = v_s$ and $\|op(\tilde{e})\| = op(v_\Sigma, \widetilde{\|e\|})$. Given an annotated term $a \in \mathfrak{A}_t$, its erasure $|a|$ produces a term with empty annotation sets, inductively defined so that $|v_s| = v_s$ and $|op(e_\Sigma, \tilde{e})| = op(\varnothing, \widetilde{|e|})$.

Machine steps. The syntax of rule stacks π is given by $\pi ::= \mathsf{init} \mid (\rho, \Sigma) :: \pi$. We denote configurations $\langle a\,;\pi \mid \tilde{e}\rangle_{\mathsf{m}}$ as *forward*, a special case being *initial* ones $\langle a\rangle_{\mathsf{zs}}$. *Backward* configurations are of the form $\langle \pi\,; a \mid \tilde{e}\rangle_{\mathsf{bm}}$ with *normal-form* ones $\langle a\rangle_{\mathsf{nf}}$ as a subcase.

Figure 6 presents the forward and backward steps generated from an inductive rule ρ, an initial rule init, and an axiom. The forward step for an inductive rule goes from the conclusion to the premise, while the backward step goes in the opposite direction. Terms are extended with the rule variable for annotated sets v_Σ. The initial rule case is the same as the inductive one but simpler, as there is no side-condition: the annotated sets of the term v_t in $\langle v_t\rangle_{\mathsf{zs}}$ are assumed to be empty. We can see that the annotations are erased after applying an axiom, as we end up with $\langle |\,\|e_t\|\,|\rangle_{\mathsf{zs}}$. There is no backward step associated to axioms.

What remains are the switching steps when we realize that the current mode m does not apply to the term $op(v_\Sigma, \tilde{v})$ we reduce. These are the "otherwise" steps in Figures 2 and 4, which actually cover different cases. The first possibility is that op does not have a rule applying to it in the mode m. For such cases, we add a step

$$\langle op(v_\Sigma, \tilde{v})\,;\pi \mid \tilde{e}\rangle_{\mathsf{m}} \mapsto \langle \pi\,; op(v_\Sigma \cup \{\phi(\mathsf{m}, \tilde{e})\}, \tilde{v}) \mid \tilde{e}\rangle_{\mathsf{bm}}$$

When going to a backward configuration, we extend the annotation set of the operator with the current annotation.

The other case is that no rule $\dfrac{e^i_t \xrightarrow{\tilde{f}_i}_{\mathsf{m}_i} v_t \qquad \mathcal{P}_i(\tilde{w})}{op(\tilde{v}) \xrightarrow{\tilde{e}}_{\mathsf{m}} v_t}\,\rho_i$ for op in the mode m applies, because either the premise or the side condition do not hold. If the machine has already checked that the premise fails, then e^i_t has been annotated with $\phi(\mathsf{m}_i, \tilde{f}_i)$. The corresponding switching step is therefore

$$\langle op(v_\Sigma, \tilde{v})\,;\pi \mid \tilde{e}\rangle_{\mathsf{m}} \mapsto \langle \pi\,; op(v_\Sigma \cup \{\phi(\mathsf{m}, \tilde{e})\}, \tilde{v}) \mid \tilde{e}\rangle_{\mathsf{bm}} \text{ if } \bigwedge_i \left(\phi(\mathsf{m}_i, \tilde{f}_i) \in \mathsf{an}(\|e^i_t\|) \vee \neg\mathcal{P}_i(\tilde{w})\right)$$

Equivalence. The equivalence between the zipper semantics and its derived NDAM is proved in Appendix D; we state here the main results. We let T (resp. A) range over (resp. annotated) ground terms. For all T, we write $\|T\|^{\varnothing}$ for the corresponding annotated term with empty annotations sets. For all A, we write $|A|$ for A where all annotations sets are made empty; there exists an unique T such that $|A| = \|T\|^{\varnothing}$. We call a *search path* a sequence of machine steps $\mapsto^{+}$ which does not go through an initial configuration. Search paths are finite, and result either in an initial or a normal-form configuration.

▸ **Theorem 8.** *For all T, there exists n such that any search path starting from $\langle \|T\|^{\varnothing} \rangle_{\mathsf{zs}}$ is of size at most n. For all maximal search paths $\langle \|T\|^{\varnothing} \rangle_{\mathsf{zs}} \mapsto^{+} c$, either $c = \langle \|T'\|^{\varnothing} \rangle_{\mathsf{zs}}$ for some T', or $c = \langle A \rangle_{\mathsf{nf}}$ for some A with $|A| = \|T\|^{\varnothing}$.*

We write $\vdash T \rightarrow_{\mathsf{zs}} T'$ when there exists a zipper semantics derivation ended with $T \rightarrow_{\mathsf{zs}} T'$. Search paths correspond to derivations in the following way.

▸ **Theorem 9.** *For all T, T', and A,*

- *$\vdash T \rightarrow_{\mathsf{zs}} T'$ iff there exists a search path $\langle \|T\|^{\varnothing} \rangle_{\mathsf{zs}} \mapsto^{+} \langle \|T'\|^{\varnothing} \rangle_{\mathsf{zs}}$;*
- *T is a normal form iff there exists a search path $\langle \|T\|^{\varnothing} \rangle_{\mathsf{zs}} \mapsto^{+} \langle A \rangle_{\mathsf{nf}}$ with $|A| = \|T\|^{\varnothing}$.*

5 Related Work

The zipper semantics of the process calculi are inspired by complementary semantics [31], a format dedicated to bisimulation proofs. In both semantics, the derivation tree of two communicating processes is sequentialized. The difference is in the transition labels, which should be as minimal as possible in complementary semantics to keep the bisimulation proofs simple, while ours are detailed enough to be able to reconstruct the whole term.

Typical abstract machines for deterministic languages based on the λ-calculus are in refocused form [10]; such machines continue term decomposition from the contraction site. They have been shown to be uniformly derivable from the underlying reduction semantics by a refocusing method [6, 43], and the correctness of the derivation hinges on the unique decomposition property. NDAMs do not have this property, and after contracting a redex they completely reconstruct the term. An optimization similar to refocusing for non-deterministic languages appears more challenging in general. Another common feature of abstract machines for the λ-calculus is an efficient implementation of substitution with environments [7]. The use of environments is orthogonal to the derivation of NDAMs: if the source zipper semantics uses environments, then so does its derived NDAM. We consider substitution-based zipper semantics in this paper because they are simpler than environment-based ones.

Process algebras have been implemented in various frameworks ranging from rewriting logic [44] to biological systems [34], including dedicated implementations and abstract machines [4, 15, 17–20, 23, 33, 36, 37, 45]. These implementations are ad-hoc and calculus-specific, and only some of them are complete [4, 18, 20, 23, 36, 37]. We believe we can handle most of these calculi in our framework in a uniform and complete way. However, the resulting implementation would be “single-threaded”, while the distribution of processes is a concern of previous machines [18], especially for calculi with localities [4, 19, 20, 23, 37]. Considering the many different models of distribution, making our machine distributed requires significantly more work, especially if we want to remain generic and complete.

Our use of backtracking evokes reversible calculi [8, 46], where one can revert communication steps, not necessarily in the order they were taken, as long as the causality between them is preserved. The concerns are different, though: in reversible calculi it is to keep enough information to track causality [29, 38], while here it is to control backtracking to avoid infinite searches. As a result, we store less information in machine configurations, but the annotations we use to prevent loops would not be typically needed in the other setting.

6 Conclusion

We present a generic design of abstract machines for non-deterministic languages. The machine looks for a redex in the term, making arbitrary choices when several paths are possible, and backtracks when it reaches a subterm which cannot reduce. The machine annotates such subterms to avoid trying them again, preventing infinite search. An NDAM is automatically derived from zipper semantics, a form of SOS in which the decomposition process of a term into a context and a redex is made explicit. The machine is sound and complete w.r.t. the zipper semantics. The derivation procedure has been implemented in OCaml [5]. The presented methodology is readily applicable to other non-deterministic calculi not shown in this paper, such as concurrent lambda calculi, with communication via channels or via futures [3, 16, 35].

An improvement of the current design would be to keep as many annotations as possible after reducing, in order to prune redundant search. Another optimization would be to find a way to manage annotations that would generically enable refocusing.

Finally, we would like to derive the zipper semantics from a more commonly used format, such as reduction semantics or SOS. An appropriate starting point should be able to express the different families of non-deterministic languages, such as concurrent λ-calculi or process calculi. A multi-hole context-based reduction semantics could be such a starting point.

References

1 Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019.

2 Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard. A functional correspondence between evaluators and abstract machines. In *Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 27-29 August 2003, Uppsala, Sweden*, pages 8–19. ACM, 2003.

3 Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco. On the concurrent computational content of intermediate logics. *Theor. Comput. Sci.*, 813:375–409, 2020.

4 Philippe Bidinger, Alan Schmitt, and Jean-Bernard Stefani. An abstract machine for the kell calculus. In Martin Steffen and Gianluigi Zavattaro, editors, *Formal Methods for Open Object-Based Distributed Systems, 7th IFIP WG 6.1 International Conference, FMOODS 2005, Athens, Greece, June 15-17, 2005, Proceedings*, volume 3535 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2005.

5 Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Non-deterministic abstract machines. Implementation available at `https://gitlab.inria.fr/skeletons/ndam/`.

6 Malgorzata Biernacka, Witold Charatonik, and Klara Zielinska. Generalized refocusing: From hybrid strategies to abstract machines. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPIcs*, pages 10:1–10:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

7 Malgorzata Biernacka and Olivier Danvy. A concrete framework for environment machines. *ACM Trans. Comput. Log.*, 9(1):6, 2007.

8 Vincent Danos and Jean Krivine. Reversible communicating systems. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 292–307. Springer, 2004.

9 Olivier Danvy. From reduction-based to reduction-free normalization. In Pieter W. M. Koopman, Rinus Plasmeijer, and S. Doaitse Swierstra, editors, *Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*, volume 5832 of *Lecture Notes in Computer Science*, pages 66–164. Springer, 2008.

10 Olivier Danvy and Lasse R. Nielsen. Syntactic theories in practice. *Electron. Notes Theor. Comput. Sci.*, 59(4):358–374, 2001.

11 Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex.* The MIT Press, 2009.

12 Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the λ-calculus. In Martin Wirsing, editor, *Formal Description of Programming Concepts - III: Proceedings of the IFIP TC 2/WG 2.2 Working Conference on Formal Description of Programming Concepts - III, Ebberup, Denmark, 25-28 August 1986*, pages 193–222. North-Holland, 1987.

13 Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992.

14 Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992.

15 Fabrice Le Fessant. *JoCaml: conception et implémentation d'un langage à agents mobiles.* PhD thesis, École polytechnique, 2001.

16 Cormac Flanagan and Matthias Felleisen. The semantics of future and an application. *J. Funct. Program.*, 9(1):1–31, 1999.

17 Cédric Fournet and Georges Gonthier. The reflexive CHAM and the join-calculus. In Hans-Juergen Boehm and Guy L. Steele Jr., editors, *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented*

at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996, pages 372–385. ACM Press, 1996.

18 Philippa Gardner, Cosimo Laneve, and Lucian Wischik. The fusion machine. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *CONCUR 2002 - Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings*, volume 2421 of *Lecture Notes in Computer Science*, pages 418–433. Springer, 2002.

19 Florence Germain, Marc Lacoste, and Jean-Bernard Stefani. An abstract machine for a higher-order distributed process calculus. *Electron. Notes Theor. Comput. Sci.*, 66(3):145–169, 2002.

20 Paola Giannini, Davide Sangiorgi, and Andrea Valente. Safe ambients: Abstract machine and distributed implementation. *Sci. Comput. Program.*, 59(3):209–249, 2006.

21 Jan Friso Groote and Frits W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Inf. Comput.*, 100(2):202–260, 1992.

22 John Hannan and Dale Miller. From operational semantics for abstract machines. *Math. Struct. Comput. Sci.*, 2(4):415–459, 1992.

23 Daniel Hirschkoff, Damien Pous, and Davide Sangiorgi. An efficient abstract machine for safe ambients. *J. Log. Algebraic Methods Program.*, 71(2):114–149, 2007.

24 Gérard P. Huet. The zipper. *J. Funct. Program.*, 7(5):549–554, 1997.

25 Simon L. Peyton Jones. Implementing lazy functional languages on stock hardware: The spineless tagless G-machine. *J. Funct. Program.*, 2(2):127–202, 1992.

26 Claude Kirchner, Radu Kopetz, and Pierre-Etienne Moreau. Anti-pattern matching. In Rocco De Nicola, editor, *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4421 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2007.

27 Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

28 Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

29 Ivan Lanese and Doriana Medic. A general approach to derive uncontrolled reversible semantics. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 33:1–33:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

30 Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 145–155. IEEE Computer Society, 2008.

31 Sergueï Lenglet, Alan Schmitt, and Jean-Bernard Stefani. Characterizing contextual equivalence in calculi with passivation. *Inf. Comput.*, 209(11):1390–1433, 2011.

32 Xavier Leroy. The ZINC experiment: an economical implementation of the ML language. Technical report 117, INRIA, 1990.

33 Luís M. B. Lopes, Fernando M. A. Silva, and Vasco Thudichum Vasconcelos. A virtual machine for a process calculus. In Gopalan Nadathur, editor, *Principles and Practice of Declarative Programming, International Conference PPDP'99, Paris, France, September 29 - October 1, 1999, Proceedings*, volume 1702 of *Lecture Notes in Computer Science*, pages 244–260. Springer, 1999.

34 Urmi Majumder and John H. Reif. Design of a biomolecular device that executes process algebra. *Nat. Comput.*, 10(1):447–466, 2011.

35 Joachim Niehren, Jan Schwinghammer, and Gert Smolka. A concurrent lambda calculus with futures. *Theor. Comput. Sci.*, 364(3):338–356, 2006.

36 Andrew Phillips and Luca Cardelli. A correct abstract machine for the stochastic pi-calculus. In *Concurrent Models in Molecular Biology*, 2004.

37 Andrew Phillips, Nobuko Yoshida, and Susan Eisenbach. A distributed abstract machine for boxed ambient calculi. In David A. Schmidt, editor, *Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2986 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2004.

38 Iain C. C. Phillips and Irek Ulidowski. Reversing algebraic process calculi. *J. Log. Algebraic Methods Program.*, 73(1-2):70–96, 2007.

39 Gordon D. Plotkin. A structural approach to operational semantics. Technical Report FN-19, DAIMI, Department of Computer Science, Aarhus University, Aarhus, Denmark, September 1981.

40 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014.

41 Davide Sangiorgi. Bisimulation in higher-order process calculi. In Ernst-Rüdiger Olderog, editor, *Programming Concepts, Methods and Calculi, Proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET '94) San Miniato, Italy, 6-10 June, 1994*, volume A-56 of *IFIP Transactions*, pages 207–224. North-Holland, 1994.

42 Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes.* Cambridge University Press, 2001.

43 Filip Sieczkowski, Malgorzata Biernacka, and Dariusz Biernacki. Automating derivations of abstract machines from reduction semantics: - A generic formalization of refocusing in Coq. In Jurriaan Hage and Marco T. Morazán, editors, *Implementation and Application of Functional Languages - 22nd International Symposium, IFL 2010, Alphen aan den Rijn, The Netherlands, September 1-3, 2010, Revised Selected Papers*, volume 6647 of *Lecture Notes in Computer Science*, pages 72–88. Springer, 2010.

44 Prasanna Thati, Koushik Sen, and Narciso Martí-Oliet. An executable specification of asynchronous pi-calculus semantics and may testing in maude 2.0. *Electron. Notes Theor. Comput. Sci.*, 71:261–281, 2002.

45 David Turner. *The Polymorphic Pi-calculus:Theory and Implementation.* PhD thesis, University of Edinburgh, 1995.

46 Irek Ulidowski, Ivan Lanese, Ulrik Pagh Schultz, and Carla Ferreira, editors. *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*, volume 12070 of *Lecture Notes in Computer Science*. Springer, 2020.

A Lambda-calculus

We prove the correspondence between the zipper and reduction semantics.

▸ **Lemma 10.** *For all* $t \xrightarrow{s,\mathbb{E}}_{\mathsf{lam}} t'$*, we have* $\mathbb{E}[t \mathbin{@} s] \to_{\mathsf{rs}} t'$*.*

For all $t \xrightarrow{\mathbb{E}}_{\mathsf{app}} t'$*, we have* $\mathbb{E}[t] \to_{\mathsf{rs}} t'$*.*

Proof. The first item is by definition, and the second one is proved by induction on the derivation of $t \xrightarrow{\mathbb{E}}_{\mathsf{app}} t'$. The base case is $\mathsf{app}\beta$, where we conclude using the first item.

For the recursive case, suppose we apply appL: we have $t \mathbin{@} s \xrightarrow{\mathbb{E}}_{\mathsf{app}} t'$ because $t \xrightarrow{@\, s \,::\, \mathbb{E}}_{\mathsf{app}} t'$. By induction, we have $(@\, s :: \mathbb{E})[t] \to_{\mathsf{rs}} t'$, i.e., $\mathbb{E}[t \mathbin{@} s] \to_{\mathsf{rs}} t'$, as wished. The other cases are similar. ◂

▸ **Theorem 11.** *For all* $t \to_{\mathsf{zs}} t'$*, we have* $t \to_{\mathsf{rs}} t'$*.*

For completeness, we want to prove that $\mathbb{E}[(\lambda x.t'') \mathbin{@} s] \to_{\mathsf{rs}} \mathbb{E}[t''\{s/x\}]$ implies $\mathbb{E}[(\lambda x.t'') \mathbin{@} s] \to_{\mathsf{zs}} \mathbb{E}[t''\{s/x\}]$. First, we notice that $\lambda x.t'' \xrightarrow{s,\mathbb{E}}_{\mathsf{lam}} \mathbb{E}[t''\{s/x\}]$ holds by definition of $\xrightarrow{s,\mathbb{E}}_{\mathsf{lam}}$. With $\mathsf{app}\beta$, we get $(\lambda x.t'') \mathbin{@} s \xrightarrow{\mathbb{E}}_{\mathsf{app}} \mathbb{E}[t''\{s/x\}]$. To conclude, we use the following result.

▸ **Lemma 12.** *For all* $t \xrightarrow{\mathbb{E}}_{\mathsf{app}} t'$*, we have* $\mathbb{E}[t] \xrightarrow{\bullet}_{\mathsf{app}} t'$*.*

Proof. We proceed by induction on $\mathbb{E}$. There is nothing to prove for $\bullet$. If $\mathbb{E} = \lambda x :: \mathbb{E}'$, then $t \xrightarrow{\lambda x \,::\, \mathbb{E}'}_{\mathsf{app}} t'$ implies $\lambda x.t \xrightarrow{\mathbb{E}'}_{\mathsf{app}} t'$ by $\mathsf{app}\lambda$, from which we deduce $\mathbb{E}'[\lambda x.t] \xrightarrow{\bullet}_{\mathsf{app}} t'$ by the induction hypothesis , i.e., $\mathbb{E}[t] \xrightarrow{\bullet}_{\mathsf{app}} t'$, as wished. The proof is similar in the remaining cases. ◂

▸ **Theorem 13.** *For all* $t \to_{\mathsf{rs}} t'$*, we have* $t \to_{\mathsf{zs}} t'$*.*

B HOcore

The output-first zipper semantics is equivalent to reduction semantics in the following way.

▸ **Lemma 14.** *For all* $R \xrightarrow{\mathbb{G},a,\mathcal{S},P,\mathbb{E},\mathbb{F}}_{\mathsf{in}} R'$*, there exists* R'' *such that either* $R' = \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[R''\{P/X\}]]$ *if* $\mathcal{S} = \mathcal{L}$ *or* $R' = \mathbb{E}[\mathbb{G}[R''\{P/X\}] \parallel \mathbb{F}[\mathbf{0}]]$ *if* $\mathcal{S} = \mathcal{R}$*.*

For all $P \xrightarrow{\mathbb{F},\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'$*, we have either* $\mathbb{E}[\mathbb{F}[P] \parallel R] \to_{\mathsf{rs}} P'$ *if* $\mathcal{S} = \mathcal{L}$ *or* $\mathbb{E}[R \parallel \mathbb{F}[P]] \to_{\mathsf{rs}} P'$ *if* $\mathcal{S} = \mathcal{R}$*.*

For all $P \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'$*, we have* $\mathbb{E}[P] \to_{\mathsf{rs}} P'$*.*

Each result is proved by induction on the zipper derivation.

▸ **Theorem 15.** *For all* $P \to_{\mathsf{zs}} P'$*, we have* $P \to_{\mathsf{rs}} P'$*.*

The reverse implication relies on the following results about contexts in zipper semantics.

▸ **Lemma 16.** *For all* $R \xrightarrow{\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}}_{\mathsf{in}} R'$*, we have* $\mathbb{G}[R] \xrightarrow{\bullet,\mathcal{S},a,P,\mathbb{E},\mathbb{F}}_{\mathsf{in}} R'$*.*

For all $P \xrightarrow{\mathbb{F},\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'$*, we have* $\mathbb{F}[P] \xrightarrow{\bullet,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'$*.*

For all $P \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'$*, we have* $\mathbb{E}[P] \xrightarrow{\bullet}_{\mathsf{par}} P'$*.*

$$\textsf{init}\quad \frac{P \xrightarrow{\bullet}_{\textsf{par}} P'}{P \rightarrow_{\textsf{zs}} P'} \qquad \textsf{parL}\quad \frac{P \xrightarrow{\parallel Q :: \mathbb{E}}_{\textsf{par}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\textsf{par}} P'}\ (s) \qquad \textsf{parCom}\quad \frac{P \xrightarrow{\bullet, \mathbb{E}, Q}_{\textsf{left}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\textsf{par}} P'}$$

$$\textsf{leftParL}\quad \frac{P \xrightarrow{\parallel Q :: \mathbb{F}, \mathbb{E}, R}_{\textsf{left}} P'}{P \parallel Q \xrightarrow{\mathbb{F}, \mathbb{E}, R}_{\textsf{left}} P'}\ (s) \qquad \textsf{leftOut}\quad \frac{R \xrightarrow{\bullet, a, P, \mathbb{E}, \mathbb{F}}_{\textsf{in}} P'}{\overline{a}\langle P\rangle \xrightarrow{\mathbb{F}, \mathbb{E}, R}_{\textsf{left}} P'} \qquad \textsf{leftIn}\quad \frac{R \xrightarrow{\bullet, a, X, P, \mathbb{E}, \mathbb{F}}_{\textsf{out}} P'}{a(X).P \xrightarrow{\mathbb{F}, \mathbb{E}, R}_{\textsf{left}} P'}$$

$$\textsf{inParL}\quad \frac{R \xrightarrow{\parallel Q :: \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}}_{\textsf{in}} P'}{R \parallel Q \xrightarrow{\mathbb{G}, a, P, \mathbb{E}, \mathbb{F}}_{\textsf{in}} P'}\ (s) \qquad \textsf{inCom}\quad \frac{}{a(X).R \xrightarrow{\mathbb{G}, a, P, \mathbb{E}, \mathbb{F}}_{\textsf{in}} \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]}$$

$$\textsf{outParL}\quad \frac{R \xrightarrow{\parallel Q :: \mathbb{G}, a, X, P, \mathbb{E}, \mathbb{F}}_{\textsf{out}} P'}{R \parallel Q \xrightarrow{\mathbb{G}, a, X, P, \mathbb{E}, \mathbb{F}}_{\textsf{out}} P'}\ (s) \qquad \textsf{outCom}\quad \frac{}{\overline{a}\langle R\rangle \xrightarrow{\mathbb{G}, a, X, P, \mathbb{E}, \mathbb{F}}_{\textsf{out}} \mathbb{E}[\mathbb{F}[P\{R/X\}] \parallel \mathbb{G}[\mathbf{0}]]}$$

Figure 7 Left-first Zipper Semantics for HOcore

Suppose $R \rightarrow_{\textsf{rs}} R'$ with $R = \mathbb{E}[\mathbb{F}[\overline{a}\langle Q\rangle] \parallel \mathbb{G}[a(X).P]]$; the proof is similar in the symmetric case. We have $a(X).P \xrightarrow{\mathbb{G}, \mathcal{L}, a, Q, \mathbb{E}, \mathbb{F}}_{\textsf{in}} R'$, and by the first item of Lemma 16, we deduce $\mathbb{G}[a(X).P] \xrightarrow{\bullet, \mathcal{L}, a, Q, \mathbb{E}, \mathbb{F}}_{\textsf{in}} R'$. We get $\overline{a}\langle Q\rangle \xrightarrow{\mathbb{F}, \mathcal{L}, \mathbb{E}, \mathbb{G}[a(X).P]}_{\textsf{out}} R'$ by rule outIn, i.e., $\mathbb{F}[\overline{a}\langle Q\rangle] \xrightarrow{\bullet, \mathcal{L}, \mathbb{E}, \mathbb{G}[a(X).P]}_{\textsf{out}} R'$ with the second item. With rule parOutL, we obtain $\mathbb{F}[\overline{a}\langle Q\rangle] \parallel \mathbb{G}[a(X).P] \xrightarrow{\mathbb{E}}_{\textsf{par}} R'$, from which we can conclude using the last item.

▸ **Theorem 17.** *For all $P \rightarrow_{\textsf{rs}} P'$, we have $P \rightarrow_{\textsf{zs}} P'$.*

The left-first semantics for HOcore is given in Figure 7. The par transition is going through the process to find the parallel composition at the root of the communication redex, building the context $\mathbb{E}$ surrounding the redex at the same time. Finding the parallel composition triggers the $\xrightarrow{\mathbb{F}, \mathbb{E}, R}_{\textsf{left}}$ transition, which looks for an input or an output in the process on the left, while building the context $\mathbb{F}$ and remembering $\mathbb{E}$ and the process on the right R. If we find an output, we look for an input on the same name in R using $\xrightarrow{\mathbb{G}, a, P, \mathbb{E}, \mathbb{F}}_{\textsf{in}}$ (rule leftOut), otherwise we look for an output using $\xrightarrow{\mathbb{G}, a, X, P, \mathbb{E}, \mathbb{F}}_{\textsf{out}}$ (rule leftIn). These two transitions are building the context $\mathbb{G}$ and use the remaining arguments to compute the results of the communication (rules inCom and outCom).

The corresponding NDAM is in Figure 8, except for the out and bout modes, which are symmetric to the in and bin modes.

C HOπ

We present the zipper semantics of HOπ, an extension of HOcore with name restriction. The main difficulty is that the evaluation contexts surrounding the communicating processes can be themselves modified by the reduction.

$$\langle P\rangle_{\mathsf{zs}} \mapsto \langle P \,;\, \mathsf{init} \mid \bullet\rangle_{\mathsf{par}}$$

$$\begin{array}{rcll}
\langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{E}\rangle_{\mathsf{par}} & \mapsto & \langle P \,;\, (\mathsf{parL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{E}\rangle_{\mathsf{par}} & \text{if } \mathsf{par} \notin \mathsf{an}(P) \\
\langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{E}\rangle_{\mathsf{par}} & \mapsto & \langle Q \,;\, (\mathsf{parR}, \Sigma) :: \pi \mid P \parallel :: \mathbb{E}\rangle_{\mathsf{par}} & \text{if } \mathsf{par} \notin \mathsf{an}(Q) \\
\langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{E}\rangle_{\mathsf{par}} & \mapsto & \langle P \,;\, (\mathsf{parCom}, \Sigma) :: \pi \mid \bullet, \mathbb{E}, Q\rangle_{\mathsf{left}} & \text{if } (\mathsf{left}, |Q|) \notin \mathsf{an}(P) \\
\langle P \,;\, \pi \mid \mathbb{E}\rangle_{\mathsf{par}} & \mapsto & \langle \pi \,;\, P^{\cup \mathsf{par}} \mid \mathbb{E}\rangle_{\mathsf{bpar}} & \text{otherwise}
\end{array}$$

$$\begin{array}{rcl}
\langle \mathsf{init} \,;\, P \mid \bullet\rangle_{\mathsf{bpar}} & \mapsto & \langle P\rangle_{\mathsf{nf}} \\
\langle (\mathsf{parL}, \Sigma) :: \pi \,;\, P \mid \parallel Q :: \mathbb{E}\rangle_{\mathsf{bpar}} & \mapsto & \langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{E}\rangle_{\mathsf{par}} \\
\langle (\mathsf{parR}, \Sigma) :: \pi \,;\, Q \mid P \parallel :: \mathbb{E}\rangle_{\mathsf{bpar}} & \mapsto & \langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{E}\rangle_{\mathsf{par}}
\end{array}$$

$$\begin{array}{rcll}
\langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}} & \mapsto & \langle P \,;\, (\mathsf{leftParL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}} & \text{if } (\mathsf{left}, |R|) \notin \mathsf{an}(P) \\
\langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}} & \mapsto & \langle Q \,;\, (\mathsf{leftParR}, \Sigma) :: \pi \mid P \parallel :: \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}} & \text{if } (\mathsf{left}, |R|) \notin \mathsf{an}(Q) \\
\langle \overline{a}^{\Sigma}\langle P\rangle \,;\, \pi \mid \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}} & \mapsto & \langle R \,;\, (\mathsf{leftOut}, \Sigma) :: \pi \mid \bullet, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \text{if } (\mathsf{in}, a) \notin \mathsf{an}(R) \\
\langle a^{\Sigma}(X).P \,;\, \pi \mid \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}} & \mapsto & \langle R \,;\, (\mathsf{leftIn}, \Sigma) :: \pi \mid \bullet, a, \mathbb{F}, \mathbb{E}, X, P\rangle_{\mathsf{out}} & \text{if } (\mathsf{out}, a) \notin \mathsf{an}(R) \\
\langle P \,;\, \pi \mid \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}} & \mapsto & \langle \pi \,;\, P^{\cup(\mathsf{left}, R)} \mid \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{bleft}} & \text{otherwise}
\end{array}$$

$$\begin{array}{rcl}
\langle (\mathsf{parCom}, \Sigma) :: \pi \,;\, P \mid \bullet, \mathbb{E}, Q\rangle_{\mathsf{bleft}} & \mapsto & \langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{E}\rangle_{\mathsf{par}} \\
\langle (\mathsf{leftParL}, \Sigma) :: \pi \,;\, P \mid \parallel Q :: \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{bleft}} & \mapsto & \langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}} \\
\langle (\mathsf{leftParR}, \Sigma) :: \pi \,;\, Q \mid P \parallel :: \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{bleft}} & \mapsto & \langle P \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}}
\end{array}$$

$$\begin{array}{rcll}
\langle R \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \mapsto & \langle R \,;\, (\mathsf{inParL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \text{if } (\mathsf{in}, a) \notin \mathsf{an}(R) \\
\langle R \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \mapsto & \langle Q \,;\, (\mathsf{inParR}, \Sigma) :: \pi \mid R \parallel :: \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \text{if } (\mathsf{in}, a) \notin \mathsf{an}(Q) \\
\langle a^{\Sigma}(X).R \,;\, \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \mapsto & \langle |\mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]|\rangle_{\mathsf{zs}} & \\
\langle R \,;\, \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} & \mapsto & \langle \pi \,;\, R^{\cup(\mathsf{in}, a)} \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{bin}} & \text{otherwise}
\end{array}$$

$$\begin{array}{rcl}
\langle (\mathsf{leftOut}, \Sigma) :: \pi \,;\, R \mid \bullet, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{bin}} & \mapsto & \langle \overline{a}^{\Sigma}\langle P\rangle \,;\, \pi \mid \mathbb{F}, \mathbb{E}, R\rangle_{\mathsf{left}} \\
\langle (\mathsf{inParL}, \Sigma) :: \pi \,;\, R \mid \parallel Q :: \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{bin}} & \mapsto & \langle R \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}} \\
\langle (\mathsf{inParR}, \Sigma) :: \pi \,;\, Q \mid R \parallel :: \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{bin}} & \mapsto & \langle R \parallel^{\Sigma} Q \,;\, \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F}\rangle_{\mathsf{in}}
\end{array}$$

Figure 8 Left-first NDAM for HOcore

C.1 Syntax and Semantics

We add name restriction to HOcore processes and frames.

$$P, Q, R ::= \dots \mid \nu a.P \qquad \mathfrak{F} ::= \dots \mid \nu a$$

To remain close to HOcore, the calculus of this section is asynchronous: outputs $\overline{a}\langle P\rangle$ do not have a continuation, unlike the original HOπ [41], Adding continuations would not be an issue as pointed out in Remark 3.

The scope of a in $\nu a.P$ is restricted to P, so that a communication on a is possible inside P only. For instance, the process $a(X).X \parallel \nu a.\overline{a}\langle \mathbf{0}\rangle$ cannot reduce, because the name a is restricted to the process on the right. In general, a process $\mathbb{E}[\overline{a}\langle P\rangle]$ or $\mathbb{E}[a(X).P]$ cannot communicate on a if $\mathbb{E}$ captures a. To check this, we compute the set of names bound by $\mathbb{E}$, written $\mathsf{bn}(\mathbb{E})$, as follows.

$$\mathsf{bn}(\bullet) \triangleq \varnothing \qquad \mathsf{bn}(\parallel P :: \mathbb{E}) \triangleq \mathsf{bn}(\mathbb{E})$$

$$\mathsf{bn}(\nu a :: \mathbb{E}) \triangleq \{a\} \cup \mathsf{bn}(\mathbb{E}) \qquad \mathsf{bn}(P \parallel :: \mathbb{E}) \triangleq \mathsf{bn}(\mathbb{E})$$

Name restriction does not forbid the communication on unrestricted names, but the scope of restricted names has to be enlarged to prevent them from escaping their delimiter. For example, we have

$$b(X).(X \parallel \overline{c}\langle \mathbf{0}\rangle) \parallel \nu a.(\overline{b}\langle a(Y).Y\rangle \parallel \overline{a}\langle \mathbf{0}\rangle) \rightarrow_{\mathsf{rs}} \\ \nu a.(a(Y).Y \parallel \overline{c}\langle \mathbf{0}\rangle \parallel \mathbf{0} \parallel \overline{a}\langle \mathbf{0}\rangle)$$

The scope of a has been extended to include the receiving process on b. This phenomenon is known as *scope extrusion*. To reflect it at the level of contexts, we define an operation $\mathsf{extr}(\mathbb{E})$ which returns a pair of contexts $(\mathbb{E}_1, \mathbb{E}_2)$ such that $\mathbb{E}_2$ contains the binding frames, while $\mathbb{E}_1$ contains the remaining frames. We assume free names to be distinct from bound names using α-conversion if necessary, to avoid capture during extrusion.

$$\mathsf{extr}(\bullet) \triangleq (\bullet, \bullet) \qquad \frac{\mathsf{extr}(\mathbb{E}) = (\mathbb{E}_1, \mathbb{E}_2)}{\mathsf{extr}(\nu a :: \mathbb{E}) \triangleq (\mathbb{E}_1, \nu a :: \mathbb{E}_2)} \qquad \frac{\mathsf{extr}(\mathbb{E}) = (\mathbb{E}_1, \mathbb{E}_2)}{\mathsf{extr}(\parallel P :: \mathbb{E}) \triangleq (\parallel P :: \mathbb{E}_1, \mathbb{E}_2)}$$

$$\frac{\mathsf{extr}(\mathbb{E}) = (\mathbb{E}_1, \mathbb{E}_2)}{\mathsf{extr}(P \parallel :: \mathbb{E}) \triangleq (P \parallel :: \mathbb{E}_1, \mathbb{E}_2)}$$

We define the reduction semantics $\rightarrow_{\mathsf{rs}}$ of HOπ as follows, assuming $a \notin \mathsf{bn}(\mathbb{F}) \cup \mathsf{bn}(\mathbb{G})$ and $\mathsf{extr}(\mathbb{F}) = (\mathbb{F}_1, \mathbb{F}_2)$.

$$\mathbb{E}[\mathbb{F}[\overline{a}\langle Q\rangle] \parallel \mathbb{G}[a(X).P]] \rightarrow_{\mathsf{rs}} \mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[P\{Q/X\}]]]$$
$$\mathbb{E}[\mathbb{G}[a(X).P] \parallel \mathbb{F}[\overline{a}\langle Q\rangle]] \rightarrow_{\mathsf{rs}} \mathbb{E}[\mathbb{F}_2[\mathbb{G}[P\{Q/X\}] \parallel \mathbb{F}_1[\mathbf{0}]]]$$

C.2 Zipper Semantics and NDAM

We present the zipper semantics of HOπ in Figure 9. The out and in transitions differ from HOcore as they carry two contexts $\mathbb{F}_1$ and $\mathbb{F}_2$: as in the reduction semantics, $\mathbb{F}_1$ collects the parallel compositions (rules $\mathsf{outParL}$ and $\mathsf{outParR}$) while $\mathbb{F}_2$ collects the name restrictions (rule outNu).

Checking that the name a on which the communication happens is not captured by $\mathbb{F}_2$ or $\mathbb{G}$ is not done the same way in the out and in transitions, because the transitions themselves

$$\textsf{init}\ \dfrac{P \xrightarrow{\bullet}_{\mathsf{par}} P'}{P \to_{\mathsf{zs}} P'} \qquad \textsf{parNu}\ \dfrac{P \xrightarrow{\nu a :: \mathbb{E}}_{\mathsf{par}} P'}{\nu a.P \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'} \qquad \textsf{parL}\ \dfrac{P \xrightarrow{\parallel Q :: \mathbb{E}}_{\mathsf{par}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'}\ (s) \qquad \textsf{parOutL}\ \dfrac{P \xrightarrow{\bullet,\bullet,\mathcal{L},\mathbb{E},Q}_{\mathsf{out}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'}$$

$$\textsf{parOutR}\ \dfrac{Q \xrightarrow{\bullet,\bullet,\mathcal{R},\mathbb{E},P}_{\mathsf{out}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'}$$

$$\textsf{outParL}\ \dfrac{P \xrightarrow{\parallel Q :: \mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'}{P \parallel Q \xrightarrow{\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'}\ (s) \qquad \textsf{outNu}\ \dfrac{P \xrightarrow{\mathbb{F}_1,\nu b :: \mathbb{F}_2,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'}{\nu b.P \xrightarrow{\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'}$$

$$\textsf{outIn}\ \dfrac{R \xrightarrow{\bullet,\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} P' \qquad a \notin \mathsf{bn}(\mathbb{F}_2)}{\overline{a}\langle P\rangle \xrightarrow{\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'}$$

$$\textsf{inParL}\ \dfrac{R \xrightarrow{\parallel Q :: \mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} P'}{R \parallel Q \xrightarrow{\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} P'}\ (s) \qquad \textsf{inNu}\ \dfrac{R \xrightarrow{\nu b :: \mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} P' \qquad a \neq b}{\nu b.R \xrightarrow{\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} P'}$$

$$\textsf{inComL}\ \dfrac{a = b}{b(X).R \xrightarrow{\mathbb{G},\mathcal{L},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} \mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]]}\ (s)$$

Figure 9 Zipper Semantics for HOπ

are not completely symmetric. In the input transition, we already know the name a, so we simply verify that the names bound by $\mathbb{G}$ differ from a on the fly in rule inNu. We cannot do the same in rule outNu, because we do yet not know a at this point. We know a when we find the output (rule outIn), so we check here that $\mathbb{F}_2$ does not capture it.

We first prove that zipper semantics implies reduction semantics.

▸ **Lemma 18.** *For all transitions $R \xrightarrow{\mathbb{G},a,\mathcal{S},P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} R'$, there exists R'' such that $R' = \mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[R''\{P/X\}]]]$ if $\mathcal{S} = \mathcal{L}$ and $R' = \mathbb{E}[\mathbb{F}_2[\mathbb{G}[R''\{P/X\}] \parallel \mathbb{F}_1[\mathbf{0}]]]$ if $\mathcal{S} = \mathcal{R}$.*

For all transitions $P \xrightarrow{\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'$ and $\mathbb{F}$ such that $\mathsf{extr}(\mathbb{F}) = (\mathbb{F}_1,\mathbb{F}_2)$, we have $\mathbb{E}[\mathbb{F}[P] \parallel R] \to_{\mathsf{rs}} P'$ if $\mathcal{S} = \mathcal{L}$ and $\mathbb{E}[R \parallel \mathbb{F}[P]] \to_{\mathsf{rs}} P'$ if $\mathcal{S} = \mathcal{R}$.

For all $P \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'$, we have $\mathbb{E}[P] \to_{\mathsf{rs}} P'$.

Proof. We sketch the proof of the second item, the others are easy. The proof is by induction on the derivation of the out transition. We assume $\mathcal{S} = \mathcal{L}$, the case $\mathcal{S} = \mathcal{R}$ is similar. In the base case (rule outIn), we have $P = \overline{a}\langle P''\rangle$ and $R \xrightarrow{\bullet,a,\mathcal{S},P'',\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} P'$, which implies $P' = \mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[R''\{P''/X\}]]]$ for some R'' by the first item.

Suppose we are in the case of rule outNu, and let $\mathbb{F}$ such that $\mathsf{extr}(\mathbb{F}) = (\mathbb{F}_1,\mathbb{F}_2)$. Then $P = \nu a.P''$ and $P'' \xrightarrow{\mathbb{F}_1,\nu a.\mathbb{F}_2,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'$. By induction, for all $\mathbb{F}'$ such that $\mathsf{extr}(\mathbb{F}') = (\mathbb{F}_1, \nu a :: \mathbb{F}_2)$, we have $\mathbb{E}[\mathbb{F}'[P''] \parallel R] \to_{\mathsf{rs}} P'$. But since $\mathsf{extr}(\mathbb{F}) = (\mathbb{F}_1,\mathbb{F}_2)$, we have $\mathsf{extr}(\nu a :: \mathbb{F}) = (\mathbb{F}_1, \nu a :: \mathbb{F}_2)$ by definition, so we can apply the induction hypothesis to obtain $\mathbb{E}[(\nu a.\mathbb{F})[P''] \parallel R] \to_{\mathsf{rs}}$

$$\langle P\rangle_{\mathsf{zs}} \mapsto \langle P\,;\mathsf{init}\,|\,\bullet\rangle_{\mathsf{par}}$$

$$\begin{aligned}
\langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}} &\mapsto \langle P\,;(\mathsf{parL},\Sigma) :: \pi\,|\,\parallel Q :: \mathbb{E}\rangle_{\mathsf{par}} && \text{if } \mathsf{par} \notin \mathsf{an}(P)\\
\langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}} &\mapsto \langle Q\,;(\mathsf{parR},\Sigma) :: \pi\,|\,P \parallel :: \mathbb{E}\rangle_{\mathsf{par}} && \text{if } \mathsf{par} \notin \mathsf{an}(Q)\\
\langle \nu^{\Sigma} a.P\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}} &\mapsto \langle P\,;(\mathsf{parNu},\Sigma) :: \pi\,|\,\nu a :: \mathbb{E}\rangle_{\mathsf{par}} && \text{if } \mathsf{par} \notin \mathsf{an}(P)\\
\langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}} &\mapsto \langle P\,;(\mathsf{parOutL},\Sigma) :: \pi\,|\,\bullet,\bullet,\mathcal{L},\mathbb{E},Q\rangle_{\mathsf{out}} && \text{if } (\mathsf{out},|Q|,\bullet) \notin \mathsf{an}(P)\\
\langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}} &\mapsto \langle Q\,;(\mathsf{parOutR},\Sigma) :: \pi\,|\,\bullet,\bullet,\mathcal{R},\mathbb{E},P\rangle_{\mathsf{out}} && \text{if } (\mathsf{out},|P|,\bullet) \notin \mathsf{an}(Q)\\
\langle P\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}} &\mapsto \langle \pi\,;P^{\cup\mathsf{par}}\,|\,\mathbb{E}\rangle_{\mathsf{bpar}} && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
\langle \mathsf{init}\,;P\,|\,\bullet\rangle_{\mathsf{bpar}} &\mapsto \langle P\rangle_{\mathsf{nf}}\\
\langle (\mathsf{parL},\Sigma) :: \pi\,;P\,|\,\parallel Q :: \mathbb{E}\rangle_{\mathsf{bpar}} &\mapsto \langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}}\\
\langle (\mathsf{parR},\Sigma) :: \pi\,;Q\,|\,P \parallel :: \mathbb{E}\rangle_{\mathsf{bpar}} &\mapsto \langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}}\\
\langle (\mathsf{parNu},\Sigma) :: \pi\,;P\,|\,\nu a :: \mathbb{E}\rangle_{\mathsf{bpar}} &\mapsto \langle \nu^{\Sigma} a.P\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}}
\end{aligned}$$

$$\begin{aligned}
\langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}} &\mapsto \langle P\,;(\mathsf{outParL},\Sigma) :: \pi\,|\,\parallel Q :: \mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}} && \text{if } (\mathsf{out},|R|,\mathbb{F}_2) \notin \mathsf{an}(P)\\
\langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}} &\mapsto \langle Q\,;(\mathsf{outParR},\Sigma) :: \pi\,|\,P \parallel :: \mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}} && \text{if } (\mathsf{out},|R|,\mathbb{F}_2) \notin \mathsf{an}(Q)\\
\langle \nu^{\Sigma} a.P\,;\pi\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}} &\mapsto \langle P\,;(\mathsf{outNu},\Sigma) :: \pi\,|\,\mathbb{F}_1,\nu a :: \mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}} && \text{if } (\mathsf{out},|R|,\nu a.\mathbb{F}_2) \notin \mathsf{an}(P)\\
\langle \overline{a}^{\Sigma}\langle P\rangle\,;\pi\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}} &\mapsto \langle R\,;(\mathsf{outIn},\Sigma) :: \pi\,|\,\bullet,\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} && \text{if } (\mathsf{in},a) \notin \mathsf{an}(R), a \notin \mathsf{bn}(\mathbb{F}_2)\\
\langle P\,;\pi\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}} &\mapsto \langle \pi\,;P^{\cup(\mathsf{out},|R|)}\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{bout}} && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
\langle (\mathsf{parOutL},\Sigma) :: \pi\,;P\,|\,\bullet,\bullet,\mathcal{L},\mathbb{E},Q\rangle_{\mathsf{bout}} &\mapsto \langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}}\\
\langle (\mathsf{parOutR},\Sigma) :: \pi\,;Q\,|\,\bullet,\bullet,\mathcal{R},\mathbb{E},P\rangle_{\mathsf{bout}} &\mapsto \langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{E}\rangle_{\mathsf{par}}\\
\langle (\mathsf{outParL},\Sigma) :: \pi\,;P\,|\,\parallel Q :: \mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{bout}} &\mapsto \langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}}\\
\langle (\mathsf{outParR},\Sigma) :: \pi\,;Q\,|\,P \parallel :: \mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{bout}} &\mapsto \langle P \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}}\\
\langle (\mathsf{outNu},\Sigma) :: \pi\,;P\,|\,\mathbb{F}_1,\nu a :: \mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{bout}} &\mapsto \langle \nu^{\Sigma} a.P\,;\pi\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}}
\end{aligned}$$

$$\begin{aligned}
\langle R \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} &\mapsto \langle R\,;(\mathsf{inParL},\Sigma) :: \pi\,|\,\parallel Q :: \mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} && \text{if } (\mathsf{in},a) \notin \mathsf{an}(R)\\
\langle R \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} &\mapsto \langle Q\,;(\mathsf{inParR},\Sigma) :: \pi\,|\,R \parallel :: \mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} && \text{if } (\mathsf{in},a) \notin \mathsf{an}(Q)\\
\langle \nu^{\Sigma} b.R\,;\pi\,|\,\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} &\mapsto \langle R\,;(\mathsf{inNu},\Sigma) :: \pi\,|\,\nu b :: \mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} && \text{if } (\mathsf{in},a) \notin \mathsf{an}(R), b \neq a\\
\langle b^{\Sigma}(X).R\,;\pi\,|\,\mathbb{G},\mathcal{L},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} &\mapsto \langle |\mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]]|\rangle_{\mathsf{zs}} && \text{if } a = b\\
\langle b^{\Sigma}(X).R\,;\pi\,|\,\mathbb{G},\mathcal{R},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} &\mapsto \langle |\mathbb{E}[\mathbb{F}_2[\mathbb{G}[R\{P/X\}] \parallel \mathbb{F}_1[\mathbf{0}]]]|\rangle_{\mathsf{zs}} && \text{if } a = b\\
\langle R\,;\pi\,|\,\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}} &\mapsto \langle \pi\,;R^{\cup(\mathsf{in},a)}\,|\,\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{bin}} && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
\langle (\mathsf{outIn},\Sigma) :: \pi\,;R\,|\,\bullet,\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{bin}} &\mapsto \langle \overline{a}^{\Sigma}\langle P\rangle\,;\pi\,|\,\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R\rangle_{\mathsf{out}}\\
\langle (\mathsf{inParL},\Sigma) :: \pi\,;R\,|\,\parallel Q :: \mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{bin}} &\mapsto \langle R \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}}\\
\langle (\mathsf{inParR},\Sigma) :: \pi\,;Q\,|\,R \parallel :: \mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{bin}} &\mapsto \langle R \parallel^{\Sigma} Q\,;\pi\,|\,\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}}\\
\langle (\mathsf{inNu},\Sigma) :: \pi\,;R\,|\,\nu b :: \mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{bin}} &\mapsto \langle \nu^{\Sigma} b.R\,;\pi\,|\,\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2\rangle_{\mathsf{in}}
\end{aligned}$$

Figure 10 Non-Deterministic Abstract Machine for HOπ

P'. This is the same as $\mathbb{E}[\mathbb{F}[\nu a.P''] \parallel R] \rightarrow_{\mathsf{rs}} P'$, but $\nu a.P'' = P$, so we get the expected result. The cases of rules $\mathsf{outParL}$ and $\mathsf{outParR}$ are similar. ◂

▸ **Theorem 19.** *For all $P \rightarrow_{\mathsf{zs}} P'$, we have $P \rightarrow_{\mathsf{rs}} P'$.*

The proof of the reverse implication follows the same strategy as in HOcore, using the following result.

▸ **Lemma 20.** *For all $R \xrightarrow{\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} R'$, we have $\mathbb{G}[R] \xrightarrow{\bullet,\mathcal{S},a,P,\mathbb{E},\mathbb{F}_1,\mathbb{F}_2}_{\mathsf{in}} R'$.*

For all $P \xrightarrow{\mathbb{F}_1,\mathbb{F}_2,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'$ and $\mathbb{F}$ such that $\mathsf{extr}(\mathbb{F}) = (\mathbb{F}_1, \mathbb{F}_2)$, *we have $\mathbb{F}[P] \xrightarrow{\bullet,\bullet,\mathcal{S},\mathbb{E},R}_{\mathsf{out}} P'$.*

For all $P \xrightarrow{\mathbb{E}}_{\mathsf{par}} P'$, we have $\mathbb{E}[P] \xrightarrow{\bullet}_{\mathsf{par}} P'$.

▸ **Theorem 21.** *For all $P \rightarrow_{\mathsf{rs}} P'$, we have $P \rightarrow_{\mathsf{zs}} P'$.*

▸ Remark 22. The zipper semantics for HOπ cannot be written in the left-first style (Remark 4) because of scope extrusion. After finding the communicating processes $P \parallel Q$, we search for an output or input in P. Because we do not know the operator in advance, we do not know if we should decompose the context surrounding it to account for scope extrusion.

While writing the zipper semantics for HOπ requires some care, the corresponding NDAM is as expected (cf. Figure 10). A difference with HOcore is the side-conditions in the outIn and inNu rules, which are added to the step. If the side-condition is not met, the “otherwise” step applies and we switch to the backward mode bout. The side-condition also makes the output mode annotation become $(\mathsf{out}, |R|, \mathbb{F}_2)$: a process $\overline{a}\langle P \rangle$ is a normal form w.r.t. output if $\mathbb{F}_2$ captures a, so being a normal form in this mode depends on $\mathbb{F}_2$.

D Correspondence Results

We show conditions for the derived machine to be sound and complete w.r.t. the zipper semantics. The intuition is that a machine run is well-bracketed, in the sense that backward steps undo the rules applied by forward steps in the reverse order. If the machine ends up applying an axiom, then we can read from the machine run the derivation tree of the corresponding transition in zipper semantics. The proofs of this section and auxiliary lemmas can be found in Appendix D.

D.1 Preliminary Notations

Let $(\mathcal{S}, \mathcal{O}, \mathcal{R})$ be a zipper semantics with annotation function ϕ. We inductively define that a sequence of rules $\rho_1, \ldots, \rho_n$ is a derivation of a statement as follows. Given an axiom $\rho = \dfrac{\mathcal{P}(\widetilde{w})}{e_t \xrightarrow{\tilde{e}}_{\mathsf{m}} e'_t}$ and a grounding substitution σ which satisfies $\mathcal{P}$, we write $\rho \vdash (e_t \xrightarrow{\tilde{e}}_{\mathsf{m}} e'_t)\sigma$.

Given a (potentially initial) rule $\rho_1 = \dfrac{e'_t \xrightarrow{\tilde{f}}_{\mathsf{m}'} v_t \qquad \mathcal{P}(\widetilde{w})}{e_t \xrightarrow{\tilde{e}}_{\mathsf{m}} v_t}$ and a grounding substitution σ which satisfies $\mathcal{P}$, we write $\rho_1, \rho_2, \ldots \rho_n \vdash (e_t \xrightarrow{\tilde{e}}_{\mathsf{m}} v_t)\sigma$ if $\rho_2, \ldots \rho_n \vdash (e'_t \xrightarrow{\tilde{f}}_{\mathsf{m}'} v_t)\sigma$.

Given a ground statement $e_t \xrightarrow{\tilde{e}}_{\mathsf{m}} e'_t$, we write $\vdash e_t \xrightarrow{\tilde{e}}_{\mathsf{m}} e'_t$ if there exist $\rho_1, \ldots \rho_n$ such that $\rho_1, \ldots, \rho_n \vdash e_t \xrightarrow{\tilde{e}}_{\mathsf{m}} e'_t$. To make the distinction between rule entities e, f—used to write rules of the zipper semantics or machine steps—and their instances in ground statements, we write the latter using capital letters E, F, and in particular we use T for ground terms. As a result, we write $\vdash T \xrightarrow{\widetilde{E}}_{\mathsf{m}} T'$ for a derivable statements from now on. Similarly, we write A for ground annotated terms in machine configurations.

Given a ground term T, we write $\|T\|^{\varnothing}$ for the ground annotated term in which all the annotation sets are empty, i.e., $\|T\|^{\varnothing} = \|T\|\{v_\Sigma \mapsto \varnothing\}$. For all A, there exists an unique T such that $|A| = \|T\|^{\varnothing}$. In statements relating the machine to the zipper semantics, we identify $|A|$ and T, writing, e.g., $\vdash |A| \xrightarrow{\widetilde{E}}_{\mathsf{m}} T'$.

We let $\mathcal{C}$, $\mathcal{F}$, $\mathcal{I}$, and $\mathcal{B}$ range over respectively all, forward, initial, and backward machine configurations. The stepping relation between configurations is written $\mapsto$, and its (reflexive and) transitive closures are written respectively $\mapsto^*$ and $\mapsto^+$. A *search path* is a sequence $\mathcal{C} \mapsto^* \mathcal{C}'$ where only $\mathcal{C}$ or $\mathcal{C}'$ may be initial configurations.

An arbitrary configuration may contain annotations inconsistent with a machine run (e.g., a λ-abstraction annotated with lam). To rule out such configurations, we define validity as follows.

▸ **Definition 23.** *A configuration $\mathcal{C}$ is valid if there exists a ground term T such that $\langle \|T\|^{\varnothing}\rangle_{\mathsf{zs}} \mapsto^* \mathcal{C}$.*

A configuration is valid if it derives from an initial configuration with a term with empty annotations. We are sure that the annotations and the stack in $\mathcal{C}$ then result from the machine itself and are well-formed. By construction, the annotation sets of a term in a valid initial configuration $\langle A\rangle_{\mathsf{zs}}$ are all empty.

D.2 Annotations

Consider the HOcore process $R \parallel \overline{a}\langle P\rangle \parallel \overline{a}\langle Q\rangle$ where R cannot do an input on a. A machine run may try first R with an input transition on a with message P, and when it fails to find the input, it annotates R with (in, a). The annotation prevents from testing R for an input on a with message Q, because we know that the success of the input transition does not depend on the message. The annotation contains enough arguments (here a) to know that a term is a normal form w.r.t. any transition with these specific arguments, and independently from the other arguments (like the message).

The next result formalizes the idea that the arguments which are not in the annotation do not matter. It says that if two instances σ and σ' of a transition $\xrightarrow{\widetilde{e}}_{\mathsf{m}}$ agrees on the annotation $\widetilde{f} \subseteq \widetilde{e}$, then a term is a normal form w.r.t. $\xrightarrow{\widetilde{e}\sigma}_{\mathsf{m}}$ iff it is a normal form w.r.t. $\xrightarrow{\widetilde{e}\sigma'}_{\mathsf{m}}$, even if σ and σ' differ on $\widetilde{e}\backslash\widetilde{f}$.

▸ **Lemma 24.** *Let m be a mode with arguments $\widetilde{e}$, and suppose $\phi(\mathsf{m}, \widetilde{e}) = (\mathsf{m}, \widetilde{f})$. For all ground term T and grounding substitutions σ, σ' such that $\widetilde{f}\sigma = \widetilde{f}\sigma'$, we have $\neg(T \xrightarrow{\widetilde{e}\sigma}_{\mathsf{m}})$ iff $\neg(T \xrightarrow{\widetilde{e}\sigma'}_{\mathsf{m}})$.*

Proof. We proceed by induction on the metric of the zipper semantics. We remind that the annotation contains the arguments which appear either in a side-condition or a premise of a rule (cf. Section 4.3). Suppose $\neg(T \xrightarrow{\widetilde{e}\sigma}_{\mathsf{m}})$, the proof is the same in the other direction.

If $\neg(T \xrightarrow{\widetilde{e}\sigma}_{\mathsf{m}})$ because the root operator of T is not parsed in m, then we also have $\neg(T \xrightarrow{\widetilde{e}\sigma'}_{\mathsf{m}})$ for the same reason. Otherwise, we have rules ρ_i parsing the operator which do not apply, either because the side condition or the premise is not satisfied for each of them. The side conditions are not satisfied also with σ', because σ and σ' agree on the annotation, which contains the variables of the side conditions.

Let e^i be the term at the source of the premise for each ρ_i. We necessarily have $e^i\sigma = e^i\sigma'$, because the variables of the arguments which occurs in e^i are in the annotation, and σ and σ' agree on the annotation. The other variables are from the term T itself, because the rules are constructive and reversible. Because $e^i\sigma = e^i\sigma'$, we can apply the induction hypothesis on each of the premises, which therefore do not hold with σ'. In the end, the rule ρ_i also fails with σ' and we have $\neg(T \xrightarrow{\widetilde{e}\sigma'}_{\mathsf{m}})$. ◂

With this property, we prove that a term in a backward configuration cannot do the transition corresponding to the machine state. By construction, the machine adds the annotation when switching to a backward configuration, so in the following lemma, we necessarily have $\phi(\mathsf{m}, \widetilde{E}) \in \mathsf{an}(A)$.

▸ **Lemma 25.** *Let $\mathcal{B} = \langle \pi \,;A \,|\, \widetilde{E} \rangle_{\mathsf{bm}}$ be a valid configuration. We have $\neg(|A| \xrightarrow{\widetilde{E}}_{\mathsf{m}})$.*

Proof. Because $\mathcal{B}$ is valid, there exists $\mathcal{I}$ with empty annotations sets such that $\mathcal{I} \mapsto^* \mathcal{B}$. The proof is by induction on the number of machine steps. There are two kinds of transition leading to a backward configuration. The first possibility is that the root operator of A is not pattern-matched in the mode m. In that case, we have directly $\neg(|A| \xrightarrow{\widetilde{E}}_{\mathsf{m}})$.

In the second case, we have rules $\rho_i = \dfrac{e^i_t \xrightarrow{\widetilde{f_i}}_{\mathsf{m}_i} v_t \qquad \mathcal{P}_i(\widetilde{w})}{op(\widetilde{v}) \xrightarrow{\widetilde{e}}_{\mathsf{m}} v_t}$ parsing the root operator op of A, and the machine has taken the default step where none of the rules apply. Let σ be a grounding substitution such that $op(v_\Sigma, \widetilde{v})\sigma = A$ and $\widetilde{e}\sigma = \widetilde{E}$. If none of the rules applies because the predicates are not satisfied, then we have directly $\neg(|A| \xrightarrow{\widetilde{E}}_{\mathsf{m}})$.

Otherwise, we have $\phi(\mathsf{m}_i, \widetilde{f_i}\sigma) \in \mathsf{an}(e^i_t\sigma)$ for some rules. Because we start from $\mathcal{I}$ with empty annotation sets, for each of such rules, the annotation has been added in the machine run before getting to $\mathcal{B}$: there exist $\mathcal{B}_i$ such that $\mathcal{I} \mapsto^* \mathcal{B}_i \mapsto^* \mathcal{B}$, $\mathcal{B}_i = \langle \pi_i \,; e^i_t\sigma \,|\, f_i\sigma_i \rangle_{\mathsf{bm}_i}$, and $\phi(\mathsf{m}_i, \widetilde{f_i}\sigma_i) = \phi(\mathsf{m}_i, \widetilde{f_i}\sigma)$. By induction, we have $\neg(|e^i_t\sigma| \xrightarrow{\widetilde{f_i\sigma_i}}_{\mathsf{m}_i})$, which implies $\neg(|e^i_t\sigma| \xrightarrow{\widetilde{f_i\sigma}}_{\mathsf{m}_i})$ by Lemma 24. As a result, the premises of the rules do not hold, so none of the rules themselves applies to A. We have $\neg(|A| \xrightarrow{\widetilde{E}}_{\mathsf{m}})$ as required. ◂

D.3 Semantics Derivation Implies Machine Run

A zipper transition can be reflected as a machine run which never backtracks, just by mimicking the derivation tree.

▸ **Lemma 26.** *If $\vdash T \xrightarrow{\widetilde{E}}_{\mathsf{m}} T'$, then for all valid configuration $\mathcal{C} = \langle A \,; \pi \,|\, \widetilde{E} \rangle_{\mathsf{m}}$ such that $|A| = T$, there exists A' such that $\mathcal{C} \mapsto^* \langle A' \rangle_{\mathsf{zs}}$ and $|A'| = T'$.*

Proof. By induction on the size of the derivation $\rho_1, \ldots, \rho_n \vdash T \xrightarrow{\widetilde{E}}_{\mathsf{m}} T'$. If $n = 1$, then we apply an axiom, and the corresponding machine step applies directly.

Otherwise, we apply the rule $\rho_1 = \dfrac{e''_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'} v_t \qquad \mathcal{P}(\widetilde{w})}{e_t \xrightarrow{\widetilde{e}}_{\mathsf{m}} v_t}$ for some σ such that $(e_t \xrightarrow{\widetilde{e}}_{\mathsf{m}} v_t)\sigma = T \xrightarrow{\widetilde{E}}_{\mathsf{m}} T'$. Let $\mathcal{C}' = \langle e''_t\sigma \,; (\rho_1, \mathsf{an}(A)) :: \pi \,|\, \widetilde{f}\sigma \rangle_{\mathsf{m}'}$. We show that we can apply the forward step corresponding to ρ_1, i.e., $\mathcal{C} \mapsto \mathcal{C}'$. The step cannot apply only if the annotation prevents it: suppose we have $\phi(\mathsf{m}', \widetilde{f}\sigma) \in \mathsf{an}(e''_t\sigma)$. Because $\mathcal{C}$ is valid, it is derived from an initial configuration $\mathcal{I}$ without annotations, so the annotation has been added in an intermediary backward configuration: there exist A'', σ', and π' such that

$\mathcal{I} \mapsto^* \langle \pi' \,;\, A'' \,|\, \widetilde{f}\sigma' \rangle_{\mathsf{bm}'} \mapsto^* \mathcal{C}$, $|A''| = e''_t\sigma$, and $\phi(\mathsf{m}', \widetilde{f}\sigma') = \phi(\mathsf{m}', \widetilde{f}\sigma)$. By Lemma 25, we have $\neg(|A''| \xrightarrow{\widetilde{f}\sigma'}_{\mathsf{m}'})$, which implies $\neg(|A''| \xrightarrow{\widetilde{f}\sigma}_{\mathsf{m}'})$ by Lemma 24. We have a contradiction, since $\vdash (e''_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'} v_t)\sigma$ holds. Therefore we have $\phi(\mathsf{m}', \widetilde{f}\sigma) \notin \mathsf{an}(e''_t\sigma)$ and $\mathcal{C} \mapsto \mathcal{C}'$.

We also have $\rho_2, \ldots, \rho_n \vdash (e''_t \xrightarrow{\widetilde{f}}_{\mathsf{m}'} v_t)\sigma$, so by induction there exists A' such that $\mathcal{C}' \mapsto^* \langle A' \rangle_{\mathsf{zs}}$ and $|A'| = T'$, from which we can conclude the proof. ◂

A direct consequence is that the machine is complete w.r.t. semantics derivations.

▸ **Theorem 27.** *For all $\vdash T \rightarrow_{\mathsf{zs}} T'$, we have $\langle \|T\|^{\varnothing} \rangle_{\mathsf{zs}} \mapsto^* \langle \|T'\|^{\varnothing} \rangle_{\mathsf{zs}}$.*

We show that the machine is also complete w.r.t. normal forms, i.e., if a term cannot reduce in the semantics, the machine ends in the machine state for normal forms $\langle \cdot \rangle_{\mathsf{nf}}$. The following lemma expresses the idea that if a term cannot reduce in a given mode, any path from a forward configuration corresponding to these term and mode goes through a backward configuration of this term and mode.

▸ **Lemma 28.** *If $\neg(T \xrightarrow{\widetilde{E}}_{\mathsf{m}})$, then for all valid configuration $\mathcal{C} = \langle A \,;\, \pi \,|\, \widetilde{E} \rangle_{\mathsf{m}}$ such that $|A| = T$, for all search path $\mathcal{C} \mapsto^* \mathcal{C}'$, there exists A' such that either $\mathcal{C}' \mapsto^* \langle \pi \,;\, A' \,|\, \widetilde{E} \rangle_{\mathsf{bm}}$ or $\langle \pi \,;\, A' \,|\, \widetilde{E} \rangle_{\mathsf{bm}} \mapsto^* \mathcal{C}'$ with $|A'| = T$.*

Proof. We proceed by induction on the metric of the zipper semantics, and distinguish two cases. If the root operator of the zipper semantics is not parsed in m, then the only machine step possible from $\mathcal{C}$ is the step to $\langle \pi \,;\, A' \,|\, \widetilde{E} \rangle_{\mathsf{bm}}$, where A' is A where the annotation set of its root operator is extended with $\phi(\mathsf{m}, \widetilde{E})$, so the result holds.

Otherwise, there exist rules $\dfrac{e^i_t \xrightarrow{\widetilde{f_i}}_{\mathsf{m}_i} v_t \qquad \mathcal{P}_i(\widetilde{w})}{op(\widetilde{v}) \xrightarrow{\widetilde{e}}_{\mathsf{m}} v_t}\ \rho_i$ parsing the root operator op of T. Let n be the number of rules for which $\mathcal{P}_i$ is satisfied and $\phi(\mathsf{m}_i, \widetilde{f_i}\sigma) \notin \mathsf{an}(e^i_t\sigma)$ for σ a grounding substitution such that $op(v_\Sigma, \widetilde{v}) = A$ and $\widetilde{e}\sigma = \widetilde{E}$; this is the number of forward configurations $\mathcal{C}'$ than can be reached in one step from $\mathcal{C}$. We prove the result by an inner induction on n. If $n = 0$, then no forward step is possible, and only the step to $\langle \pi \,;\, A' \,|\, \widetilde{E} \rangle_{\mathsf{bm}}$ can be done, where A' is as above.

Otherwise, we can make a forward step $\langle A \,;\, \pi \,|\, \widetilde{E} \rangle_{\mathsf{m}} \mapsto \langle e^j_t\sigma \,;\, (\rho_j, \mathsf{an}(A)) :: \pi \,|\, \widetilde{f_j}\sigma \rangle_{\mathsf{m}'_j}$ for some j. However, we have $\neg(|e^j_t\sigma| \xrightarrow{\widetilde{f_j}\sigma}_{\mathsf{m}'})$, otherwise T could do a m-transition. Because $\xrightarrow{f_j}_{\mathsf{m}'}$ is smaller than $\xrightarrow{\widetilde{e}}_{\mathsf{m}}$ according to the zipper semantics metric, we can apply the outermost induction hypothesis. There exists A' such that $\langle e^j_t\sigma \,;\, (\rho_j, \mathsf{an}(A)) :: \pi \,|\, \widetilde{f_j}\sigma \rangle_{\mathsf{m}'_j} \mapsto^* \langle (\rho_j, \mathsf{an}(A)) :: \pi \,;\, A' \,|\, \widetilde{f_j}\sigma \rangle_{\mathsf{bm}'_j}$ and $|A'| = |e^j_t\sigma|$. By construction, the only possible next step is to undo ρ_j, so that $\langle (\rho_j, \mathsf{an}(A)) :: \pi \,;\, A' \,|\, \widetilde{f_j}\sigma \rangle_{\mathsf{bm}'_j} \mapsto \langle A'' \,;\, \pi \,|\, \widetilde{E} \rangle_{\mathsf{m}}$, where A'' differs from A only in their annotations sets. In particular, A'' can no longer do the step corresponding to ρ_j, so $n - 1$ configurations are reachable from A''. Therefore, we can conclude using the induction hypothesis on n. ◂

As a result, the only possible outcome for a machine run starting from a normal form is a normal-form configuration which cannot step further.

▸ **Theorem 29.** *If $\neg(T \rightarrow_{\mathsf{zs}})$, then any machine run from $\langle \|T\|^{\varnothing} \rangle_{\mathsf{zs}}$ ends with $\langle A \rangle_{\mathsf{nf}}$ such that $|A| = T$.*

The machine goes through a normal form to annotate each of its constructors from which a transition could trigger. Different runs produce the same annotations for each constructor, but generated in a different order, depending on the arbitrary choices the machine makes.

D.4 Machine Run Implies Semantics Derivation

To show that a machine run encodes a derivation, we formalize the intuition that a machine run is well-bracketed: backward steps undo the rules applied by forward steps in the reverse order. To this end, we label a forward step $\mathcal{F} \xmapsto{\rho} \mathcal{F}'$ and backward step $\mathcal{B} \xmapsto{-\rho} \mathcal{F}$ with the rule ρ it applies or unapplies, and a step switching from forward to backward with τ: $\mathcal{F} \xmapsto{\tau} \mathcal{B}$. We let λ range over these labels.

We define the backtrack-free closure $\xmapsto{\rho}_{\mathsf{bf}}$ to forget about the rules that have been applied and then unapplied in a sequence of machine steps. We write $\xmapsto{\lambda_1.\lambda_2\ldots\lambda_n}_{\mathsf{bf}}$ for a sequence $\xmapsto{\lambda_1}_{\mathsf{bf}}\xmapsto{\lambda_2}_{\mathsf{bf}} \ldots \xmapsto{\lambda_n}_{\mathsf{bf}}$. The $\mapsto_{\mathsf{bf}}$ relation is defined as follows:

$$\frac{\mathcal{C} \xmapsto{\lambda} \mathcal{C}'}{\mathcal{C} \xmapsto{\lambda}_{\mathsf{bf}} \mathcal{C}'} \qquad \frac{\mathcal{F} \xmapsto{\rho.\tau.-\rho}_{\mathsf{bf}} \mathcal{F}'}{\mathcal{F} \xmapsto{\tau}_{\mathsf{bf}} \mathcal{F}'} \qquad \frac{\mathcal{F} \xmapsto{\tau.\tau}_{\mathsf{bf}} \mathcal{F}'}{\mathcal{F} \xmapsto{\tau}_{\mathsf{bf}} \mathcal{F}'} \qquad \frac{\mathcal{F} \xmapsto{\tau.\rho}_{\mathsf{bf}} \mathcal{F}'}{\mathcal{F} \xmapsto{\rho}_{\mathsf{bf}} \mathcal{F}'}$$

Backward-free steps extend regular steps with a new behavior, as we can have a $\xmapsto{\tau}_{\mathsf{bf}}$ step between forward configurations. In such a case, the two configurations are equal up to annotations: the resulting configuration contains strictly more annotations than the source one. The other backtrack-free steps corresponds to regular machine steps.

▸ **Lemma 30.** *A step $\xmapsto{\rho}_{\mathsf{bf}}$ is between two forward configurations, a step $\xmapsto{-\rho}_{\mathsf{bf}}$ is between a backward and a forward configuration, and $\xmapsto{\tau}_{\mathsf{bf}}$ is either between a forward and a backward configuration, or two forward configurations.*

Proof. By induction on the derivation of $\xmapsto{\lambda}_{\mathsf{bf}}$. ◂

Given two configuration $\mathcal{C}_1$ and $\mathcal{C}_2$, we write $|\mathcal{C}_1| = |\mathcal{C}_2|$ if they are equal up to their terms under focus A_1 and A_2, for which we have $|A_1| = |A_2|$.

▸ **Lemma 31.** *For all $\mathcal{F} \xmapsto{\tau}_{\mathsf{bf}} \mathcal{F}'$, we have $|\mathcal{F}| = |\mathcal{F}'|$. For any other transition $\mathcal{C} \xmapsto{\lambda}_{\mathsf{bf}} \mathcal{C}'$, there exists $\mathcal{C}''$ such that $\mathcal{C} \xmapsto{\lambda} \mathcal{C}''$ and $|\mathcal{C}''| = |\mathcal{C}'|$.*

Proof. We proceed by induction on the derivation of $\mathcal{C} \xmapsto{\lambda}_{\mathsf{bf}} \mathcal{C}'$. The base case is straightforward. If $\mathcal{F} \xmapsto{\rho}_{\mathsf{bf}}\xmapsto{\tau}_{\mathsf{bf}}\xmapsto{-\rho}_{\mathsf{bf}} \mathcal{F}'$, then by the induction hypothesis, there exists $\mathcal{F}_1$, $\mathcal{B}_2$, and $\mathcal{C}''$ such that $\mathcal{F} \xmapsto{\rho} \mathcal{F}_1 \xmapsto{\tau} \mathcal{B}_2 \xmapsto{-\rho} \mathcal{C}''$ and $|\mathcal{C}''| = |\mathcal{F}'|$. The first step applies ρ which is then unapplied, so one can check that $|\mathcal{C}''| = |\mathcal{F}|$, and therefore $|\mathcal{F}| = |\mathcal{F}'|$ as required.

The case $\mathcal{F} \xmapsto{\tau}_{\mathsf{bf}}\xmapsto{\tau}_{\mathsf{bf}} \mathcal{F}'$ is easy by induction. If $\mathcal{F} \xmapsto{\tau}_{\mathsf{bf}} \mathcal{F}_0 \xmapsto{\rho}_{\mathsf{bf}} \mathcal{F}'$, then by the induction hypothesis $|\mathcal{F}| = |\mathcal{F}_0|$ and $\mathcal{F}_0 \xmapsto{\rho} \mathcal{F}''$ for some $\mathcal{F}''$ such that $|\mathcal{F}'| = |\mathcal{F}''|$. If $\mathcal{F}_0$ is able to do a $\xmapsto{\rho}$ step, then so is $\mathcal{F}$, since $|\mathcal{F}| = |\mathcal{F}_0|$ and $\mathcal{F}_0$ contains more annotations than $\mathcal{F}$. As a result, we have $\mathcal{F} \xmapsto{\rho} \mathcal{F}''$ with $|\mathcal{F}'| = |\mathcal{F}''|$, as wished. ◂

A machine run can be represented as a sequence of backtrack-free steps. Given a configuration $\mathcal{C}$, we write $\mathsf{stack}(\mathcal{C})$ for its rules stack.

▸ **Lemma 32.** *Let $\mathcal{I}$ be a valid initial configuration. For all $\mathcal{I} \mapsto^* \xmapsto{\rho} \mathcal{C}$, there exist $\rho_1 \ldots \rho_n$ such that $\mathcal{I} \xmapsto{\rho_1\ldots\rho_n}_{\mathsf{bf}} \mathcal{C}$.*

For all $\mathcal{I} \mapsto^* \xmapsto{\lambda} \mathcal{C}$ such that $\lambda \neq \rho$, there exist $\mathcal{F}$, $\rho_1 \ldots \rho_n$ such that $\mathcal{I} \xmapsto{\rho_1\ldots\rho_n}_{\mathsf{bf}} \mathcal{F} \xmapsto{\tau}_{\mathsf{bf}} \mathcal{C}$ and $\mathsf{stack}(\mathcal{F}) = \mathsf{stack}(\mathcal{C})$.

The condition on stacks in the second case implies that a backtracking step after the τ step will undo the last rule applied before $\mathcal{F}$, i.e., ρ_n. If a machine run ends with an axiom application, we are in the first case, and we get a backtrack-free sequence which corresponds to a derivation tree.

▸ **Lemma 33.** *Let $\langle A\,;\pi\mid\widetilde{E}\rangle_{\mathsf{m}}$ be a valid configuration such that there exist $\rho_1\ldots\rho_n$, A' so that $\langle A\,;\pi\mid\widetilde{E}\rangle_{\mathsf{m}} \xmapsto{\rho_1\ldots\rho_n}_{\mathsf{bf}} \langle A'\rangle_{\mathsf{zs}}$ is a search path. Then $\rho_1\ldots\rho_n \vdash |A| \xrightarrow{\widetilde{E}}_{\mathsf{m}} |A'|$.*

Proof. We proceed by induction on n. If $n = 1$, we apply an axiom. By Lemma 31, we have $\langle A\,;\pi\mid\widetilde{E}\rangle_{\mathsf{m}} \xmapsto{\rho_1} \langle A''\rangle_{\mathsf{zs}}$ for some A'' such that $|A''| = |A'|$. An instance of a machine step implies an instance of the axiom, therefore we have $\rho_1 \vdash |A| \xrightarrow{\widetilde{E}}_{\mathsf{m}} |A'|$.

If $n > 1$, we apply the induction hypothesis on the sequence $\rho_2\ldots\rho_n$ to get $\rho_2\ldots\rho_n \vdash |A''| \xrightarrow{\widetilde{E}}_{\mathsf{m}} |A'|$ for some A''. By Lemma 31, we have $\langle A\,;\pi\mid\widetilde{E}\rangle_{\mathsf{m}} \xmapsto{\rho_1} \langle A'''\,;(\rho_1,\Sigma)::\pi\mid\widetilde{F}\rangle_{\mathsf{m}'}$ where A'' is such that $|A'''| = |A''|$. Let σ be the grounding substitution of this machine step, and v_T be the rule variable designating the outcome of the transitions in the source and premise of ρ_1. We have an instance of ρ_1 by considering σ' which maps any $w \neq v_T$ to $|w\sigma|$ and v_T to $|A'|$. This instance completes the derivation and we have $\rho_1\ldots\rho_n \vdash |A| \xrightarrow{\widetilde{E}}_{\mathsf{m}} |A'|$, as wished. ◂

▸ **Theorem 34.** *For all search path $\langle \|T\|^{\varnothing}\rangle_{\mathsf{zs}} \mapsto^{+} \langle A'\rangle_{\mathsf{zs}}$, we have $\vdash T \to_{\mathsf{zs}} |A'|$.*

The machine is also sound w.r.t. normal forms: if a run ends with $\langle A\rangle_{\mathsf{nf}}$, then $|A|$ is indeed a normal form. It is a direct consequence of Lemma 25, by considering the initial mode.

▸ **Theorem 35.** *Let $\langle A\rangle_{\mathsf{nf}}$ be a valid configuration; then $|A|$ is a normal form.*

Theorems 34 and 35 show that a run reaching an initial or a normal-form configuration rightfully corresponds to a zipper derivation or lack thereof. The next result states these are the only two outcomes, and they are exclusive. We show that any machine run eventually reaches an initial or normal-form configuration. If it is possible to get to any of the reducts of a term T when we start from $\langle \|T\|^{\varnothing}\rangle_{\mathsf{zs}}$ (Theorem 27), it is no longer the case once the machine has done some non-deterministic choice. For instance in HOcore, if the machine starts exploring $P \parallel Q$ with P which contains a redex, then Q will be ignored. The machine backtracks only to undo choices that lead to a dead end, not the ones that eventually lead to a redex.

Given a partial run $\mathcal{F} \mapsto^{*} \mathcal{F}'$, either we can reach a redex from $\mathcal{F}'$, or we need to backtrack. In the latter case, the machine backtracks as little as possible, i.e., to the first configuration from which we can find a redex. We formalize this idea in the next lemma, using backtrack-free closure.

▸ **Lemma 36.** *Let $\mathcal{F} = \langle A\,;\pi\mid\widetilde{E}\rangle_{\mathsf{m}}$ be a valid configuration such that $\mathcal{F} \xmapsto{\rho_1\ldots\rho_n}_{\mathsf{bf}} \mathcal{F}'$ and $|A| \xrightarrow{\widetilde{E}}_{\mathsf{m}} T'$ for some T'.*

There exists $0 \leqslant k \leqslant n$ such that for all $T', \rho'_{k+1}\ldots\rho'_m$ verifying $\rho_1\ldots\rho_k, \rho'_{k+1}\ldots\rho_m \vdash |A| \xrightarrow{\widetilde{E}}_{m} T'$, there exists A' such that $\mathcal{F} \xmapsto{\rho_1\ldots\rho_k,\rho'_{k+1}\ldots\rho_m}_{\mathsf{bf}} \langle A'\rangle_{\mathsf{zs}}$ and $|A'| = T'$. Besides, for all $k < k' \leqslant n$, for all T', $\rho'_{k'+1}\ldots\rho'_m$, we have $\neg(\rho_1\ldots\rho_{k'}\rho'_{k'+1}\ldots\rho'_m \vdash |A| \xrightarrow{\widetilde{E}}_{\mathsf{m}} T')$.

Proof. Consider all the derivations proving a transition $\vdash |A| \xrightarrow{\widetilde{E}}_{\mathsf{m}} T'$, and take k as the length of the longest common prefix of $\rho_1\ldots\rho_n$ with these derivations—we may have $k = 0$ if a bad choice is made from the start. $\mathcal{F}_k = \langle A_k\,;\pi_k\mid\widetilde{f_k}\rangle_{\mathsf{m}_k}$ be the configuration such that $\mathcal{F} \xmapsto{\rho_1\ldots\rho_k}_{\mathsf{bf}} \mathcal{F}_k$. We show that the machine can backtrack from $\mathcal{F}'$ to a configuration equal to $\mathcal{F}_k$ up to annotations.

If $k = n$, then $\mathcal{F}_k = \mathcal{F}'$ and we do not need to backtrack. Suppose $k < n$, and let $\mathcal{F}_n = \langle A_n\,;\pi_n\mid\widetilde{f_n}\rangle_{\mathsf{m}_n}$. We necessarily have $\neg(|A_n| \xrightarrow{\widetilde{f_n}}_{\mathsf{m}_n})$, otherwise we would have $k = n$.

Therefore, by Lemma 28, there exists A'_n such that $\mathcal{F}_n \mapsto^* \langle \pi_n \,;\, A'_n \mid \widetilde{f_n} \rangle_{\mathsf{bm}_n}$ and $|A'_n| = |A_n|$. This configuration then backtracks to a configuration equal to $\mathcal{F}_{n-1}$ up to annotations. By induction, we show that we can reach $\mathcal{F}_k$. Then from this configuration, we can reach any T' verifying $\vdash |A_k| \xrightarrow{\widetilde{f_k}}_{\mathsf{m}_k} T'$ by Lemma 26, hence the result holds. ◀

▸ **Theorem 37.** *Let $\mathcal{F}$ be a valid configuration. Either $\mathcal{F} \mapsto^* \mathcal{I}$ for some $\mathcal{I}$, or $\mathcal{F} \mapsto^* \langle A \rangle_{\mathsf{nf}}$ for some A.*

Finally, search paths are finite. We cannot have an infinite sequence of $\overset{\rho}{\mapsto}_{\mathsf{bf}}$ steps, because the well-founded hypothesis on the zipper semantics carries over to these steps. We cannot have an infinite sequence of $\overset{\tau}{\mapsto}_{\mathsf{bf}}$ steps, because each of them adds at least one annotation, and the number of annotations is bounded for a given term (see Lemma 38 in the appendix).

▸ **Lemma 38.** *For all $\mathcal{I}$, there exists a finite set of annotations such that for all search path $\mathcal{I} \mapsto^* \mathcal{C}$, the annotations occurring in $\mathcal{C}$ are in this set.*

Proof. A *partial transition*, denoted by p, is a transition without its result of the form $T \xrightarrow{\tilde{E}}_{\mathsf{m}}$. We define the set S_T of partial transitions generated from T as follows:

- $T \rightarrow_{\mathsf{zs}} \in S_T$;
- for any $p \in S$, for any rule $\rho = \dfrac{e'_t \xrightarrow{\tilde{f}}_{\mathsf{m}'} v_t \qquad \mathcal{P}(\tilde{w})}{e_t \xrightarrow{\tilde{e}}_{\mathsf{m}} v_t}$, for any grounding σ satisfying $\mathcal{P}$ such that $(e_t \xrightarrow{\tilde{e}}_{\mathsf{m}})\sigma = p$, we have $(e'_t \xrightarrow{\tilde{f}}_{\mathsf{m}'})\sigma \in S_T$.

The machine explores S_T until it reaches an axiom finishing the transition, and the annotations generated during the search are built from the the partial transitions in S_T (cf. Section 4.3).

Let $\mathcal{I} = \langle \|T\|^{\varnothing} \rangle_{\mathsf{zs}}$. We prove that S_T is finite. At each step of the process, we add a finite number of partial transitions, because the number of rules is finite and the number of suitable σ is finite when we restrict them to $\mathsf{rv}(e_t \xrightarrow{\tilde{e}}_{\mathsf{m}})$. For each of such σ corresponds exactly one added premise, because the semantics is machine constructive (cf. Definition 5). The process itself cannot go on indefinitely, because the premises are strictly smaller than the conclusion according to the well-foundedness hypothesis (Definition 7). ◀

▸ **Theorem 39.** *For all $\mathcal{I}$, there exists a number n such that any search path $\mathcal{I} \mapsto^* \ldots$ has length at most n.*

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399